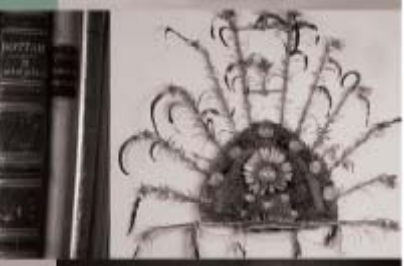
Manual

collectionconnection



collectionconnection v. 2.0

CollectionConnection makes publishing your collection easy! Electronic publishing combined with powerful full text search is only a few mouse clicks away. With minimal effort, your collection information can be integrated into your website, published on CD-Rom, or made available to Google.

With this manual, you will learn:

- all functionality of CollectionConnection.
- how to map many kinds of data into a coherent XML structure (for example, databases, PDF files, MS-Office files).
- how information from different sources can be integrated to one entry point.
- how to use the powerful and efficient searching capabilities.
- how your collection database can be published in many different ways, for example, in a website, on CD, with SOAP, or with Z39.50.
- how everything can be secured with digital signatures and encrypted exchange of information.

CollectionConnection

User Manual

Version 2.0

Copyright © 2004 VanWezel Informatiesystemen. CollectionConnection is a registered trademark. All rights reserved.

Chapter overview

1 Introduction 3

2 Getting started 9

3 Overall structure 21

4 Installation 27

5 CollectionConnection Indexer 31

6 CollectionConnection Server 79

7 CollectionConnection Distributed Server 115

8 CollectionConnection Client 123

9 Active Server Pages 133

10 CollectionConnection Reporting 139

11 CollectionConnection Imageserver 147

12 Tools 159

Appendix A Scripting Reference 177

Appendix B Copyright notices and disclaimers 185

Index 187

Table of Contents

1 Introduction 3

1.1 Introduction 3
1.2 Overview of characteristics 3
1.3 System requirements 5

2 Getting started 9

2.1 Introduction 9
2.2 Installing CollectionConnection 9
2.3 Getting the contents of a database in a website in 19 mouseclicks 9
2.4 Searching in Microsoft Word files on your harddisk 13

3 Overall structure 21

4 Installation 27

5 CollectionConnection Indexer 31

5.1 Introduction 31
5.2 Creating a new profile 31
5.3 Loading and saving a profile 31
5.4 The Log window 33
5.5 The Settings window 33
5.6 Data source elements 34
5.6.1 Introduction 34
5.6.2 Scripting 35
5.6.3 Combine indexes 37
5.6.4 Batch process index creation 38
5.6.5 ADO/Oracle specific settings 39
5.6.6 Adlib specific settings 53
5.6.7 Files specific settings 54
5.6.8 Indexing structured files 58
5.6.9 Indexing Outlook Email messages 61
5.6.10 Indexing Outlook Express Email messages 62
5.6.11 Tags 64
5.6.12 Stopwords 73
5.6.13 Dynamic Content Links 74

6 CollectionConnection Server 79

6.1 Introduction 79
6.2 The system 80
6.2.1 Creating a new profile 80
6.2.2 Loading and saving a profile 81
6.2.3 The Log window 82
6.2.4 The Settings window 83
6.2.5 General settings 83

6.2.6 Remote Log Server....84
6.2.7 CollectionConnection....85
6.2.8 HTTP/XML....86
6.2.9 Soap....87
6.2.10 HTTP/HTML-ASP....87
6.2.11 Open Archives Initiative (OAI)....90
6.2.12 Image server....91
6.2.13 Web crawler gateway....93
6.2.14 Z39.50....96
6.2.15 Index transmission....97
6.3 Query parameters....98
6.3.1 Digital signatures....99
6.3.2 Collections....99
6.3.3 Thesaurus....100
6.3.4 Spelling alternatives....101
6.3.5 Index data structure....102
6.3.6 Full text search....103
6.3.7 Authentication....107
6.3.8 Distrubuted search....108
6.3.9 Output....108
6.4 Distributing indices with the server....110

7 CollectionConnection Distributed Server....115
7.1 Introduction....115
7.2 Scalability....116
7.3 Input....117
7.4 Output....118
7.4.1 Publisher....118
7.4.2 State....119
7.5 Settings....119

8 CollectionConnection Client....123
8.1 Introduction....123
8.2 Generic variables for all objects....123
8.3 ObjectCollections....123
8.4 ObjectCategories....124
8.5 ObjectSet....124
8.5.1 Resultset....124
8.5.2 Summarization....125
8.5.3 Tags....126
8.6 Spelling....126
8.7 ThighlightSearchWords....127
8.8 ImageSize....127
8.9 ImageServerProxy....128
8.10 Security....128
8.11 Mail....128

9 Active Server Pages....133
9.1 Introduction....133

9.2 Basic retrieval method 133
9.3 Interaction process 133
9.4 Structure in the included website 135

10 CollectionConnection Reporting 139
10.1 Introduction 139
10.2 Creating reports 139
10.3 Designing reports 140

11 CollectionConnection Imageserver 147
11.1 Introduction 147
11.2 Imageserver shell integration 147
11.2.1 Image conversion 147
11.2.2 Image preview 149
11.3 Image retrieval parameters 149
11.3.1 Scaling 150
11.3.2 Border 150
11.3.3 Passepartout 150
11.3.4 Overlay image 150
11.3.5 Overlay text 151
11.3.6 Miscellaneous 151
11.4 Zoom applet 152
11.5 Image maps 153
11.6 Image proxy 154

12 Tools 159
12.1 Introduction 159
12.2 Remote Control 159
12.2.1 Remote Control Server 159
12.2.2 Remote Control Client 161
12.3 XML Viewer 165
12.4 Database Transfer 166
12.5 Image map editor 168

Appendix A Scripting Reference 177
A.1 Basic syntax 177
A.2 Script structure 177
A.3 Identifiers 177
A.4 Assign statements 178
A.5 Character strings 178
A.6 Comments 178
A.7 Variables 178
A.8 Indexes 179
A.9 Arrays 179
A.10 If statements 179
A.11 while statements 179
A.12 loop statements 180
A.13 for statements 180

A.14 select case statements....181
A.15 function and sub declaration....181
A.16 available predefined functions....182

Appendix B Copyright notices and disclaimers....185

Index....187

Introduction

1 Introduction

1.1 Introduction

CollectionConnection provides a safe and transparent way to search and browse electronic collections on intranets or the internet. Typical uses of *CollectionConnection* are:

- Integrate information from different sources to publish them as one, for example, the collection database, the library database, and electronic publications.
- Publish the collection in a large number of ways (for example, in a website, on CD-Rom, via the Open Archives Initiative, or via SOAP).
- Show collection related information on kiosks.
- Make the collection database searchable by internet search engines such as Google.
- Cooperate with other institutions to provide a webportal with common collection data.
- Include meta-information about objects or object collections in internet and intranet publications.
- Information exchange, for example using the CIMI / Dublin-Core XML specification.

1.2 Overview of characteristics

Important characteristics of *CollectionConnection* are:

- *CollectionConnection* provides an integral solution for electronic publishing of collections.
- *CollectionConnection* does not rely on an individual database or collection management system. *CollectionConnection* can use different systems and multiple data formats (e.g., The Museum System, Adlib, MS-Access, SQL-server, Oracle, XML, Microsoft Word files, PDF files, HTML Files, Outlook and Outlook Express email messages, and text files). Furthermore, *CollectionConnection* can integrate information from different sources to one entry point.
- Electronic Publications can be published in several ways, for example Internet, CD, or Kiosk.
- By using multiple ways to get to the information, simple as well as complex projects can be easily realized.
- *CollectionConnection* is compliant with many standards (among others, Open Archives Initiative, Dublin Core, Z39.50, SSL, XML-DSIG).
- *CollectionConnection* provides efficient and powerful searching capabilities.
- *CollectionConnection* has an integrated solution for images.
- *CollectionConnection* operates completely separate from the production environment, thereby providing complete safety.

Simplicity where it belongs

CollectionConnection is developed for publishing collections. It is especially helpful for publishing knowledge and information about our cultural heritage by providing support for thesauri, collections, linking objects to related information such as authors and collectors, and it has extensive and integrated support for images.

CollectionConnection hides the technical complexity of publishing as much as possible. Content providers (for example curators or librarians) work in their own environment, and with a push on the button the information from the production system is integrated in the publishing environment. Publishing on the internet can be done by yourself but can also be outsourced to your reseller. Although end-users are shielded from the technical details, they are available for developers to create advanced information retrieval solutions.

CollectionConnection is complete
All components that are needed are supplied. Publishing is fast and easy: choose the information that must be published (by mapping the source data to an XML-structure), and determine how you want to distribute the publication (internet, CD, kiosk). A publication template is part of CollectionConnection, but it can easily be adapted or replaced.

Independent of the source
CollectionConnection is not a by-product of a collection management system. It is especially developed to be able to work with multiple sources of information, e.g., dedicated systems such as The Museum System and Adlib, databases such as Microsoft Access, SQL-Server, and Oracle, or even plain XML, Microsoft Word, Microsoft Excel, or PDF files. Furthermore, a scripting language provides a powerful interface to manipulate the underlying source data.

Interoperability
CollectionConnection provides support for several publication environments:

- Your collection or a part of your collection on the internet.
- Your collection or a part of your collection on CD.
- Object information on kiosks (e.g., in a museum).
- Combining information sources (e.g., collection, library, thesaurus, electronic publications).
- With *CollectionConnection*, you can participate in collective dissemination of collections.

Powerfull development tool for electronic publishing
The information in *CollectionConnection* can be accessed in many ways: from simple and easy to complex and powerful. A layered structure provides simplicity without hindering more demanding applications:

- An integrated webserver with support for Active Server Pages (ASP) provides an easily accessible development platform.
- *CollectionConnection* can be accessed from almost all Windows development platforms via a supplied OCX-component.

- *CollectionConnection* can be accessed from SOAP, providing distributed and cross platform access.
- Data can be retrieved by a simple HTTP request.
- Programs can talk to *CollectionConnection* directly with a dedicated protocol.

Safe
The safety of your production environment is crucial. Therefore, *CollectionConnection* is not accessing your production environment directly when publishing information. Instead, there is a two stage process. First, you decide what parts of your data may be published. Second, this data is put in a separate file that can be distributed separately from your production environment.

Communication with *CollectionConnection* can take place via SSL. Then, the communication between the user and *CollectionConnection* can not be observed by outsiders. Also, access can be restricted with a username and password. Furthermore, published information can be supplemented with a digital signature. With the signature, the receiver of the information can check whether the information has been tampered with or not.

Conformance to international standards
CollectionConnection conforms to several international standards that are used by organizations in the cultural heritage domain. Data is stored as XML. Using this, the records can conform to the Dublin Core standard. Information can be retrieved with the OAI protocol (the Open Archives Initiative). *CollectionConnection* can be used by bibliographic clients that use Z39.50.

Advanced and powerful full text searching
Search facilities are elaborate. Phrase searches and searching with wildcards are no problem, even in large collections. Even complex queries will give the results in less than one second. *CollectionConnection* also provides functionality for users that do not know what they are searching for by adding structure to the search process. By providing spelling alternatives (Boedha, boedda, boedah, boeddha, bouddha), searching and browsing a thesaurus, structuring in collection sets, and dynamic content linking, it becomes easier to find information that someone is searching for.

Integrated solution for images
Images must often be edited before publishing: the image must be scaled, a border must be put around the image, it needs a background color, and possibly a copyright watermark. Such editing operations are time consuming, and must be done for each kind of publication. *CollectionConnection* solves this by doing it only when necessary: at the moment that an image is requested and shown. One source image suffices for all electronic publications. Developers of publications do not need to search for images and they do not have to edit them themselves. They only need to determine how the image should look in the context of the publication, and *CollectionConnection* takes care of the rest. Furthermore, HTML imagemaps can be created easily and shown in any size with *CollectionConnection*.

1.3 System requirements

CollectionConnection runs on Windows 98, Windows NT, Windows 2000, Windows XP, and Windows Server 2003. Necessary are:

- Source data (ADO, Adlib, XML, MS-Word files, MS-Excel files, etc.).
- Microsoft Windows Scripting Host for the server when using the ASP functionality.
- Microsoft Data components (MDAC).
- Internet Explorer 4.01 or later.

The indexer, the server, and the client can be installed on different machines (see also chapter 3). The only requirement is that the server and client can communicate by TCP/IP. The client must be installed on the machine that the website is on.

2

Getting started

2 Getting started

2.1 Introduction

In this chapter we will show some of the basic functionality of *CollectionConnection*. We show how it can be installed, and two examples will illustrate the process of creating an index, starting the server, and searching through the data. The examples use preconfigured profiles. The other chapters describe how to create new profiles and how to change existing profiles. The examples presume that the default installation options were not altered during the setup procedure.

2.2 Installing CollectionConnection

CollectionConnection can be installed by executing the file `CollectionConnection v2.0 Install.exe`. *CollectionConnection* follows a conventional installation procedure. The folder in which *CollectionConnection* will be installed and the start menu folder are configurable. The setup procedure can be halted at any time by pressing the `Cancel` button.

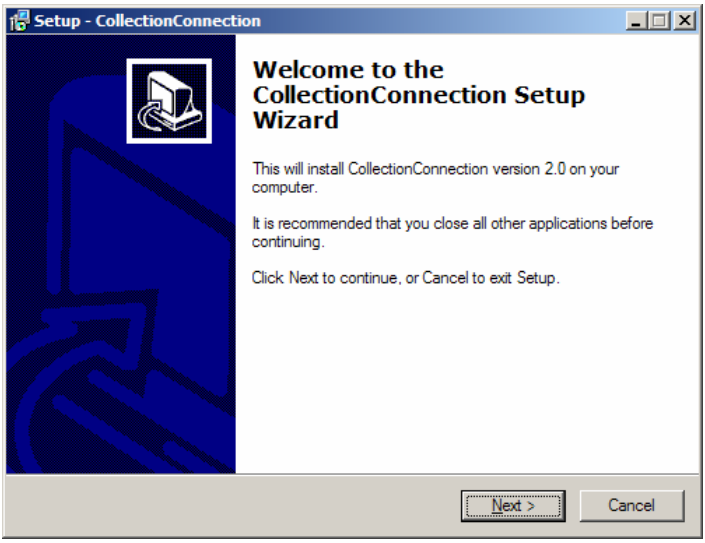

Setup - CollectionConnection
Welcome to the CollectionConnection Setup Wizard
This will install CollectionConnection version 2.0 on your computer.
It is recommended that you close all other applications before continuing.
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel

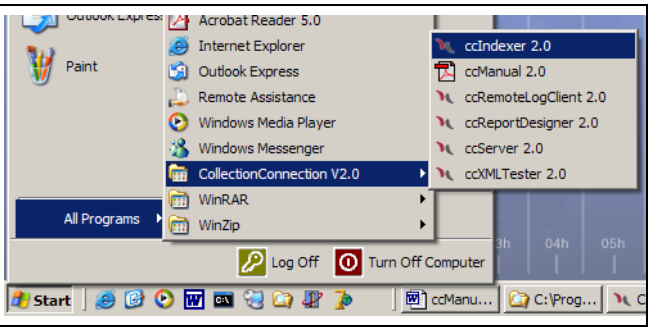
Figure 2.1. Setup wizard welcome screen

2.3 Getting the contents of a database in a website in 19 mouseclicks

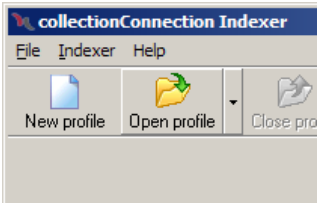
A demo database has been included in the setup: `C:\Program Files\CollectionConnection v2.0\indexer\cctestdb.mdb`. In this example, a part of the contents of this database will be put in a *CollectionConnection* index, and you will be able to search this index using the included website template. Preconfigured profiles for this database have been included during installation: `databaseExample.indexerprofile` and `databaseExample.serverprofile`.

Be careful, the *CollectionConnection* server is a true server. This means that if it is active when your computer is connected to the Internet, and you don't have a firewall, others might be able to search your index like you can yourself. This is useful when you want to publish your information on a webserver, but probably not when you are working on your home computer with an ADSL internet connection. In the included examples, remote access to the *CollectionConnection* server is restricted with the `bind to localhost` setting in the server (see section 6.2.5 on page 84), so that other computers do not have access. However, changing this will enable such access.

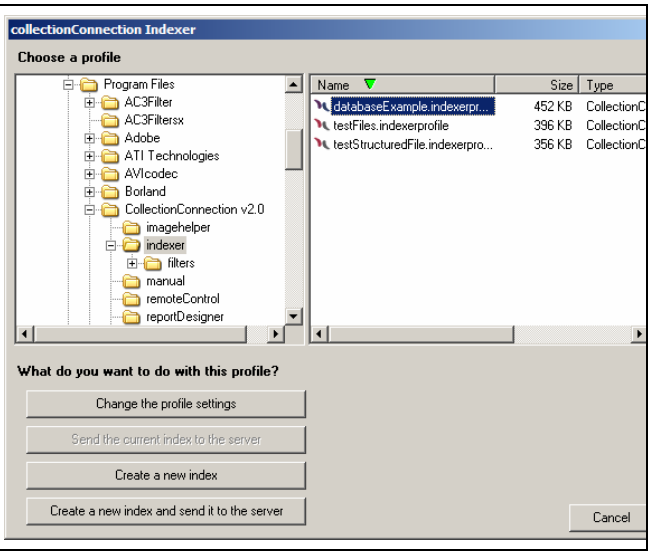
Step 1
Start the CollectionConnection Indexer from the Windows Start menu.



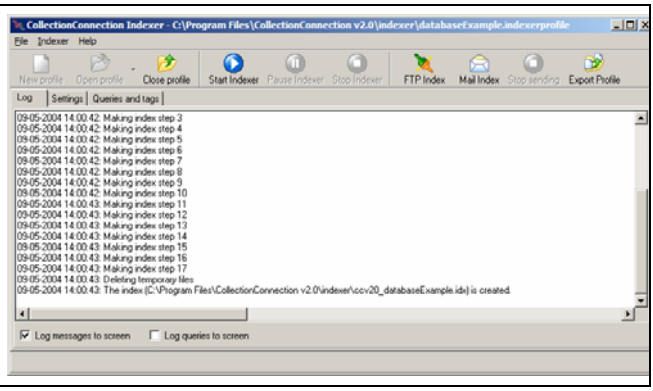
Step 2
Click the `Open profile` button.



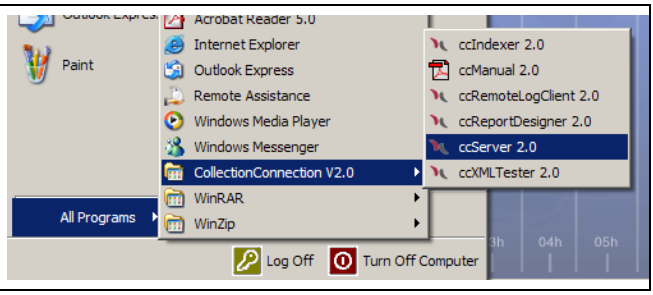
Step 3
Choose `databaseExample.indexerprofile` and click on the `Create a new index` button.



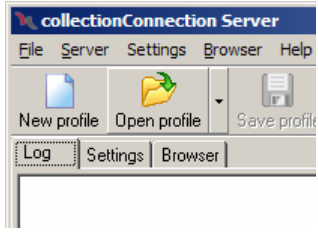
Step 4
Wait until the Indexer is finished. You can now close the Indexer.


CollectionConnection Indexer - C:\Program Files\CollectionConnection v2.0\indexer\databaseExample.indexerprofile
File Indexer Help
New profile
Open profile
Close profile
Start Indexer
Pause Indexer
Stop Indexer
FTP Index
Mail Index
Stop sending
Export Profile
Log
Settings
Queries and tags
09-05-2004 14:00:42: Making index step 3
09-05-2004 14:00:42: Making index step 4
09-05-2004 14:00:42: Making index step 5
09-05-2004 14:00:42: Making index step 6
09-05-2004 14:00:42: Making index step 7
09-05-2004 14:00:42: Making index step 8
09-05-2004 14:00:42: Making index step 9
09-05-2004 14:00:42: Making index step 10
09-05-2004 14:00:43: Making index step 11
09-05-2004 14:00:43: Making index step 12
09-05-2004 14:00:43: Making index step 13
09-05-2004 14:00:43: Making index step 14
09-05-2004 14:00:43: Making index step 15
09-05-2004 14:00:43: Making index step 16
09-05-2004 14:00:43: Making index step 17
09-05-2004 14:00:43: Deleting temporary files
09-05-2004 14:00:43: The index (C:\Program Files\CollectionConnection v2.0\indexer\ccv20_databaseExample.idx) is created.
Log messages to screen
Log queries to screen

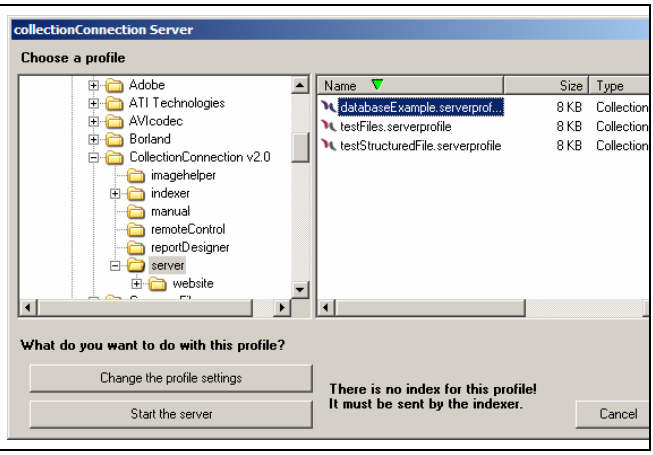
Step 5
Start the CollectionConnection Server from the Windows Start menu.



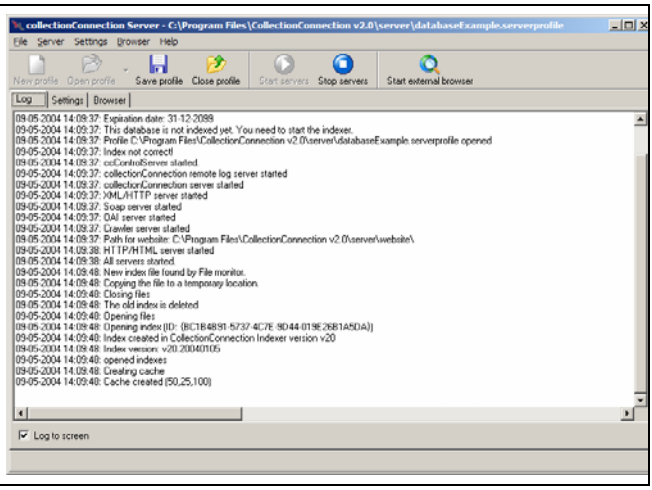
Step 6
Click the `Open profile` button.



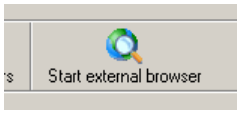
Step 7
Choose `databaseExample.serverprofile` and click on the `Start the Server` button. You can disregard the message that there is no index yet.



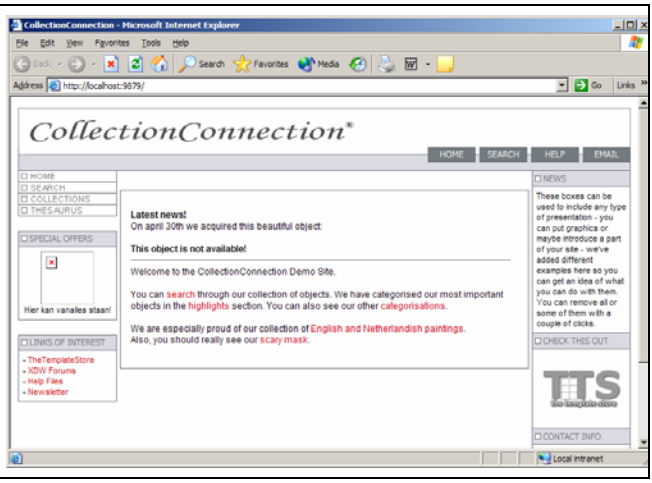
Step 8
After approximately 10 seconds, the Server will automatically copy the index that was just created by the Indexer.


collectionConnection Server - C:\Program Files\CollectionConnection v2.0\server\databaseExample.serverprofile
File Server Settings Browser Help
New profile Open profile Save profile Close profile Start servers Stop servers Start external browser
Log Settings Browser
09-05-2004 14:09:37: Expiration date: 31-12-2099
09-05-2004 14:09:37: This database is not indexed yet. You need to start the indexer.
09-05-2004 14:09:37: Profile C:\Program Files\CollectionConnection v2.0\server\databaseExample.serverprofile opened
09-05-2004 14:09:37: Index not correct!
09-05-2004 14:09:37: ccControlServer started
09-05-2004 14:09:37: collectionConnection remote log server started
09-05-2004 14:09:37: collectionConnection server started
09-05-2004 14:09:37: XML/HTTP server started
09-05-2004 14:09:37: Soap server started
09-05-2004 14:09:37: OAI server started
09-05-2004 14:09:37: Crawler server started
09-05-2004 14:09:37: Path for website: C:\Program Files\CollectionConnection v2.0\server\website\
09-05-2004 14:09:38: HTTP/HTML server started
09-05-2004 14:09:38: All servers started
09-05-2004 14:09:48: New index file found by File monitor.
09-05-2004 14:09:48: Copying the file to a temporary location.
09-05-2004 14:09:48: Closing files
09-05-2004 14:09:48: The old index is deleted
09-05-2004 14:09:48: Opening files
09-05-2004 14:09:48: Opening index (ID: {BC184891-5737-4C7E-9D44-019E26B1A5DA})
09-05-2004 14:09:48: Index created in CollectionConnection Indexer version v20
09-05-2004 14:09:48: Index version: v20.20040105
09-05-2004 14:09:48: opened indexes
09-05-2004 14:09:48: Creating cache
09-05-2004 14:09:48: Cache created (50,25,100)
Log to screen

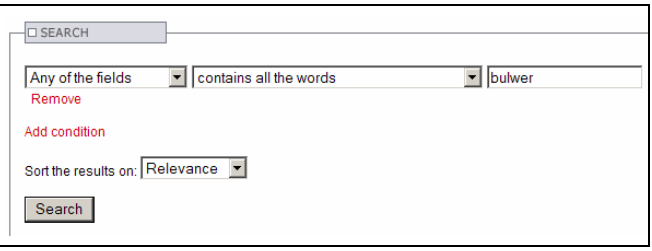
Step 9
Push the `Start external browser` button.

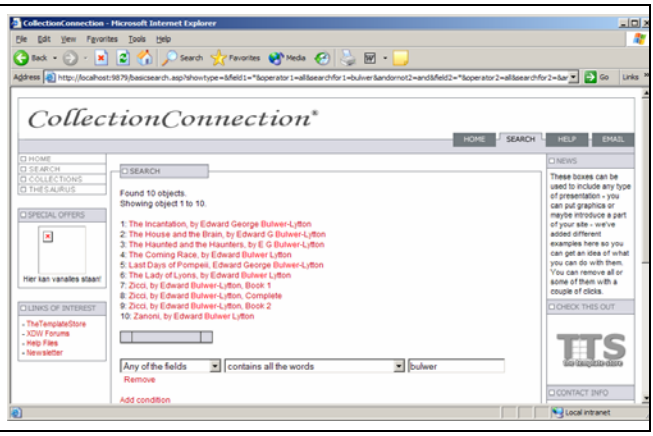


Step 10
Press the `Search` button on the left button bar.


CollectionConnection - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:9879/
CollectionConnection®
HOME SEARCH HELP EMAIL
HOME
SEARCH
COLLECTIONS
THESAURUS
SPECIAL OFFERS
Hier kan vanalles staan!
LINKS OF INTEREST
TheTemplateStore
XDW Forums
Help Files
Newsletter
Latest news!
On april 30th we acquired this beautiful object:
This object is not available!
Welcome to the CollectionConnection Demo Site.
You can search through our collection of objects. We have categorised our most important objects in the highlights section. You can also see our other categorisations.
We are especially proud of our collection of English and Netherlandish paintings.
Also, you should really see our scary mask.
NEWS
These boxes can be used to include any type of presentation - you can put graphics or maybe introduce a part of your site - we've added different examples here so you can get an idea of what you can do with them. You can remove all or some of them with a couple of clicks.
CHECK THIS OUT
CONTACT INFO
Local intranet

Step 11
Type `bulwer` in the input box and then press the `Search` button.

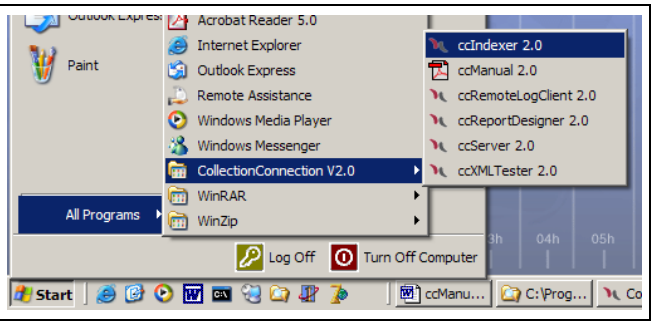


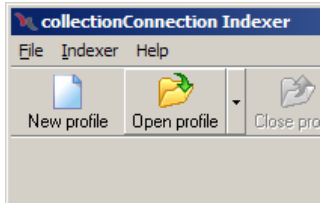
Step 12 The browser shows the results.	

You can use the created index on any computer that has *CollectionConnection* installed. The source database is not needed to search through the index. Just copy the files `databaseExample.serverprofile` and `ccv20_databaseExample.idx` to the *CollectionConnection* server folder of that computer, and open the server profile. Furthermore, you can easily create a setup program that not only contains the *CollectionConnection* server but also the index that was just created and the website template that you are using (see section 6.4 on page 110). You can now ship a searchable version of your database with a configurable front-end to anyone you want.

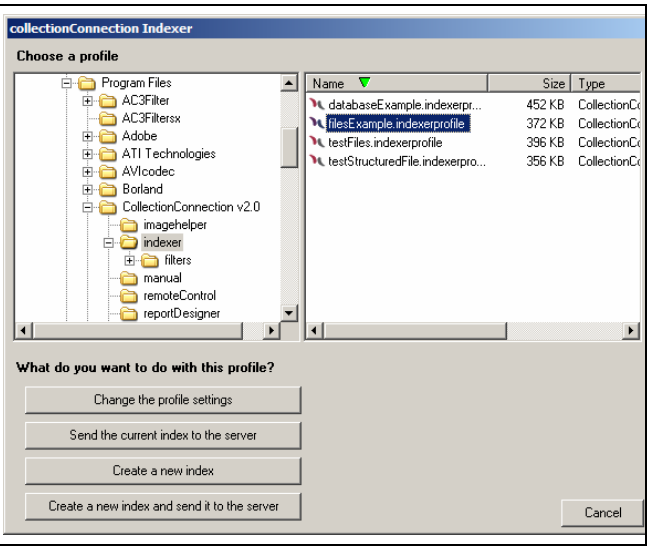
2.4 Searching in Microsoft Word files on your harddisk

In this example, you will create a *CollectionConnection* index that contains the contents of Microsoft Word files that you have on your harddisk. You will be able to search through these files from the included website template. For this, two profiles have been included: `filesExample.indexerprofile` and `filesExample.serverprofile`.

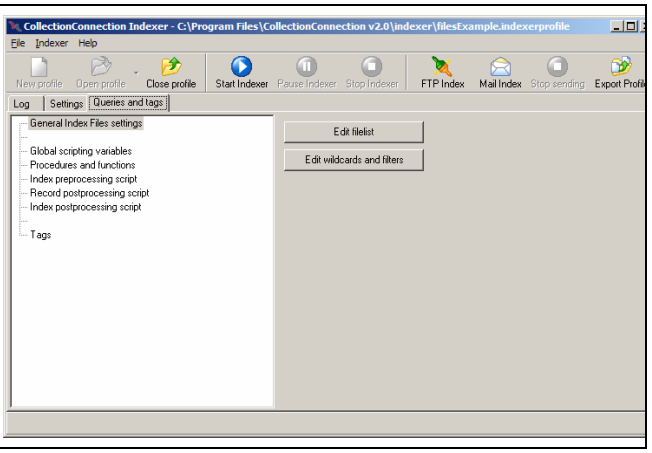
Step 1 Start the CollectionConnection Indexer from the Windows Start menu.	

Step 2 Click the `Open profile` button.	

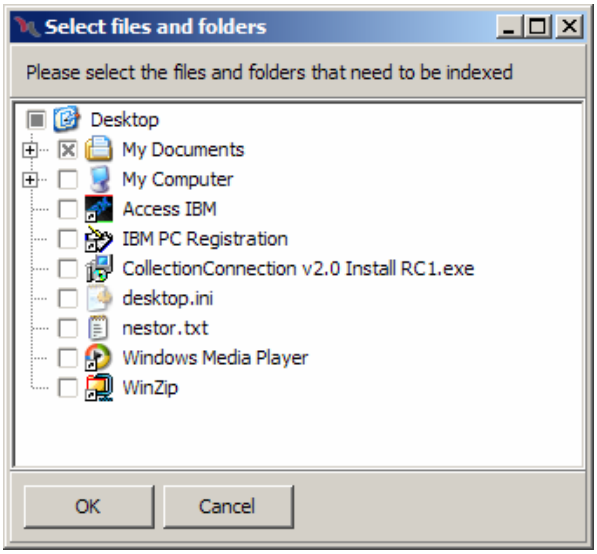
Step 3
Choose `filesExample.indexerprofile` and click on the `Change the profile settings` button.



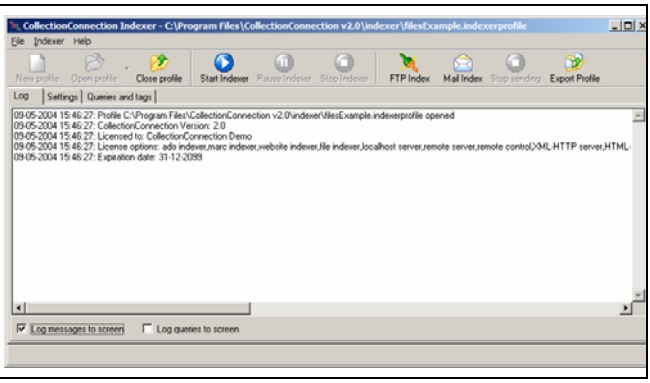
Step 4
Go to the `Queries and Tags` pane and click on the `Edit filelist` button.



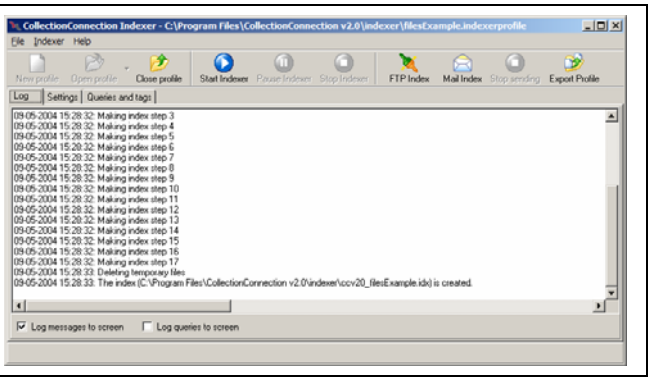
Step 5
Select the folders from which CollectionConnection should index the Word documents. By default, this will be the `My Documents` folder. When you are finished, click the `OK` button.



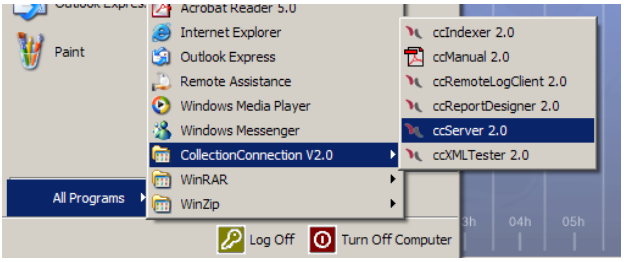
Step 6
Go to the `log` pane and click on the `Start Indexer` button.



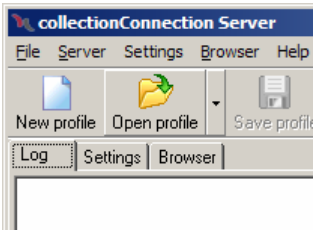
Step 7
Wait until the Indexer is finished. You can now close the Indexer.


CollectionConnection Indexer - C:\Program Files\CollectionConnection v2.0\indexer\filesExample.indexerprofile
File Indexer Help
New profile
Open profile
Close profile
Start Indexer
Pause Indexer
Stop Indexer
FTP Index
Mail Index
Stop sending
Export Profile
Log
Settings
Queries and tags
09-05-2004 15:28:32: Making index step 3
09-05-2004 15:28:32: Making index step 4
09-05-2004 15:28:32: Making index step 5
09-05-2004 15:28:32: Making index step 6
09-05-2004 15:28:32: Making index step 7
09-05-2004 15:28:32: Making index step 8
09-05-2004 15:28:32: Making index step 9
09-05-2004 15:28:32: Making index step 10
09-05-2004 15:28:32: Making index step 11
09-05-2004 15:28:32: Making index step 12
09-05-2004 15:28:32: Making index step 13
09-05-2004 15:28:32: Making index step 14
09-05-2004 15:28:32: Making index step 15
09-05-2004 15:28:32: Making index step 16
09-05-2004 15:28:32: Making index step 17
09-05-2004 15:28:33: Deleting temporary files
09-05-2004 15:28:33: The index (C:\Program Files\CollectionConnection v2.0\indexer\ccv20_filesExample.idx) is created.
Log messages to screen
Log queries to screen

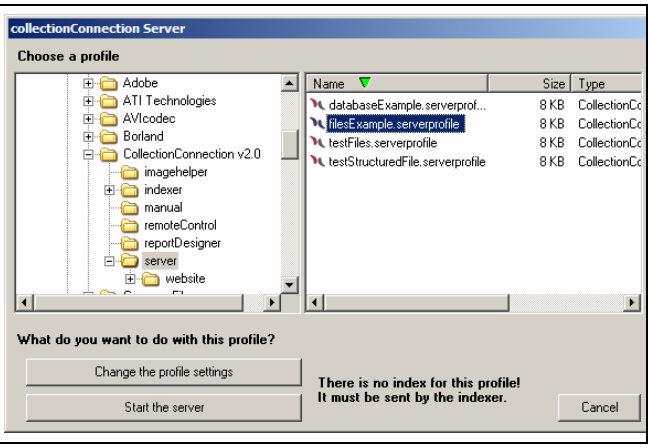
Step 8
Start the CollectionConnection Server from the Windows Start menu.



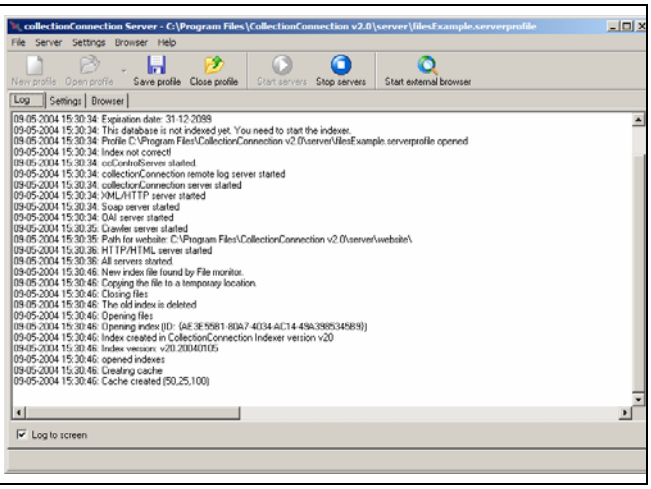
Step 9
Click the `Open profile` button.



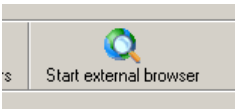
Step 10
Choose `filesExample.serverprofile` and click on the `Start the Server` button. You can disregard the message that there is no index yet.



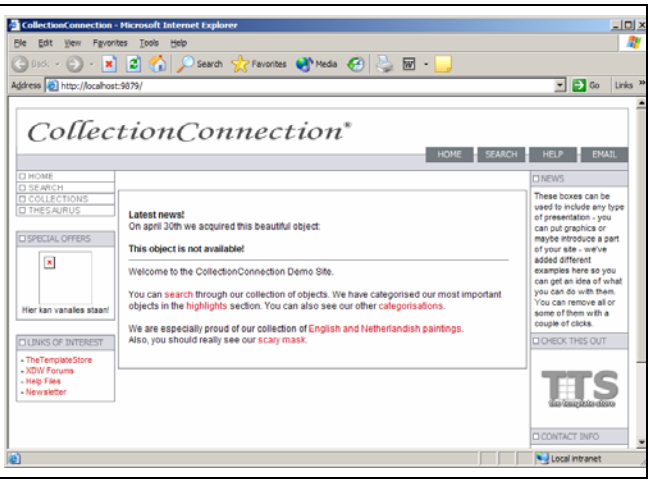
Step 11
After approximately 10 seconds, the Server will automatically copy the index that was just created by the Indexer.



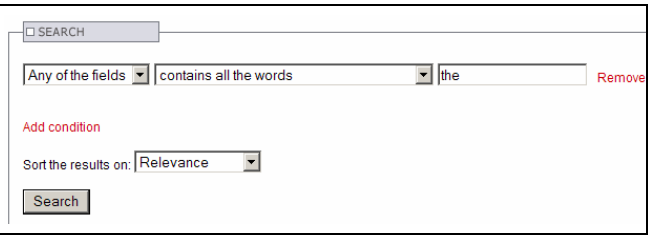
Step 12
Push the `Start external browser` button.



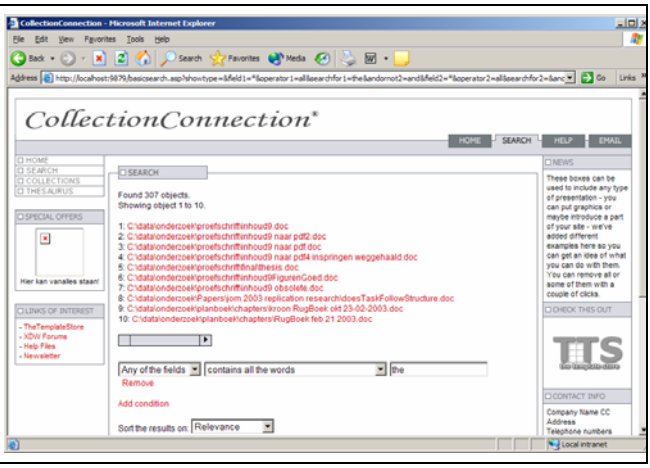
Step 13
Press the `Search` button on the left button bar.



Step 14
Type `the` (or another word that you expect to be in your documents) in the input box and then press the `Search` button.



Step 15
The browser shows the results.



In order to keep it small, the index that was just created does not include the text of the Word files nor the Word files themselves. However, this can easily be accomplished by changing a few settings in the Indexer profile. Then you can create a website in which users not only can search the files but they can download them as well; everything

(including the source files) is put in one index file for easy distribution. You can distribute this on CD, but you can put it on a webserver as well. Furthermore, you are not restricted to Microsoft Word files. PDF files, Microsoft Excel files, Microsoft Access files, text files, etcetera, can all be indexed. Additionally, it doesn't matter if they are compressed in ZIP files; *CollectionConnection* can even expand the ZIP files and index the contents of the files in the ZIP archive. How all this works, and much more, is explained in the following chapters.

3

Overall Structure

3 Overall structure

CollectionConnection provides a safe and transparent way to search and browse electronic collections on intranets or the internet. Basically, there are two steps in content disclosure: (1) choosing what data should be published, and (2) determining how others should access the data. For both steps, many possibilities and options are available, and *CollectionConnection* contains a number of tools that not only provide the necessary functionality but also to make the process of data mapping and content publishing as uncomplicated as possible. The architecture of the system consists of six parts:

1. **Indexer**: The indexer creates an index file of the database that can be searched with high efficiency.
2. **Server**: The server searches through the index file. The server can be approached by several protocols which allows access from many development environments.
3. **Client**: The client is a Microsoft Windows OCX-control that communicates with open procedure calls with the Server. The OCX can be used in Windows programming environments such as Active Server Pages (ASP), Delphi, Visual Basic, and Visual-C.
4. **Website**: The website is a of collection HTML-pages and ASP scripts that together provide internet connectivity for the collection. The ASP scripts use the Client to communicate with the Server. The appearance of the pages is easily adaptable.
5. **Reporting**: Reports are documents that contain results of search queries. They can be visually designed, and saved as, for example, Adobe Acrobat PDF files and Microsoft Word RTF files.
6. **Imaging tools**: The image server makes it easy to host the images, and it removes the need to change images before they can be published.
7. **Database transfer**: The index file can be exported to a relational database.

The following figure shows the structure of these components (the core components are bold-faced):

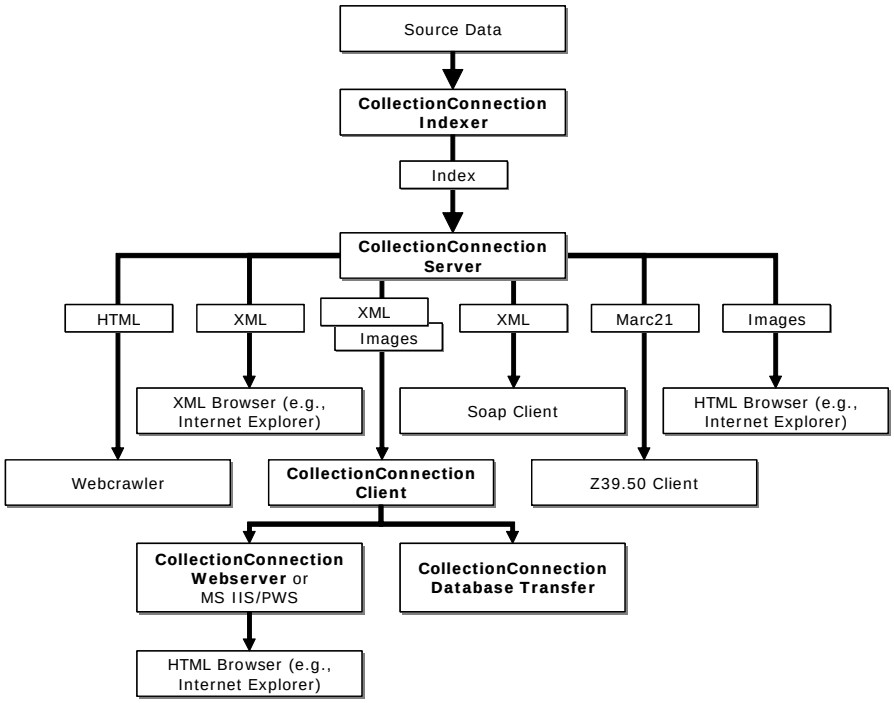


Figure 3.1. Structure of *CollectionConnection*

CollectionConnection uses XML to store the data. The Indexer contains a mapping tool to specify what sources should make up the XML contents. The Server can perform full text searches on the XML data with high efficiency. The output of the Server is again XML, but only the part that is requested. The Client contains functionality to easily get the relevant parts of the data in the XML. Figure 3.2 contains an example of the flow of information in *CollectionConnection*.

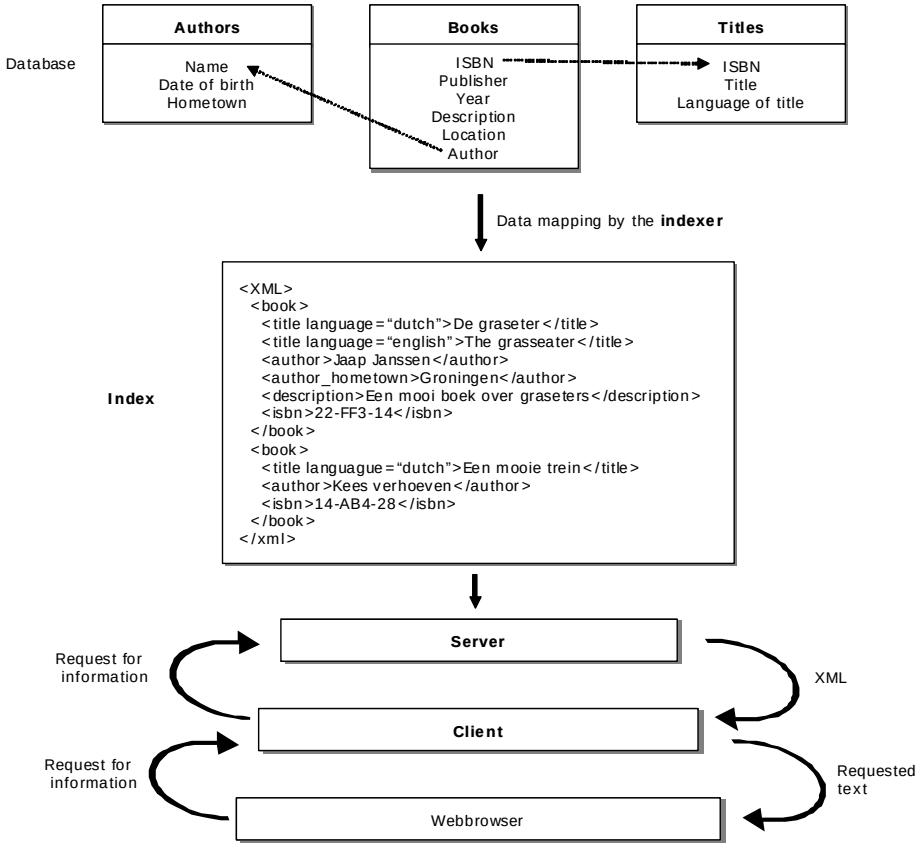


Figure 3.2. Process flow of index creation to data publishing

CollectionConnection not only contains XML-records. Structuring the records in meaningful groups is also supported in two ways:

1. Objects can be grouped in collections. The nature of a collection is not relevant for *CollectionConnection*. They can be objects that are acquired in the same year, objects that are stored in the same room, objects that are from the same era, etc. It is possible to browse through the collections, to see what objects are in a collection, and to see what collections an object is in.
2. *CollectionConnection* has support for a hierarchical structure of concepts or terms, commonly referred to as a thesaurus. Users can browse through the thesaurus itself, they can see what objects belong to an element in the thesaurus, and they can see what concepts in the thesaurus the object belongs to. This makes it possible for users to find related objects in a structured way.

Installation

4 Installation

The *CollectionConnection* programs are installed by double clicking the installation file. The program directory will contain the following:

- A subdirectory `Manual`, with the file `ccManualV2.pdf`. That's this document. To view the manual, Adobe Acrobat Reader should be installed.
- A subdirectory `Indexer`. This directory contains the Indexer program, two sample indexer profiles, and a sample database. Furthermore, it contains a subdirectory Filters that contain the filters to index files.
- A subdirectory `Server`. This directory contains the Server programs, the Client activeX, the Webserver activeX, two sample server profiles, and some miscellaneous tools. Furthermore, it contains a subdirectory `Website` where the default website is stored.
- A subdirectory `ReportDesigner`. It contains the program to design reports.
- A subdirectory `Imagehelper`. It contains programs to manage *CollectionConnection* Images.
- A subdirectory `RemoteControl`. In contains programs to remotely control another computer. The remote control probrams can be used for remote administration or for support.
- A subdirectory `cc2db`. It contains the program to transfer a *CollectionConnection* index to a relational database structure.
- A subdirectory `ImageMapEditor`. It contains a program to make HTML imagemaps.

During installation, some activeX components are registered, and icons are added to the Microsoft Windows start-programs tree.

The subdirectory `indexer\filters\Ifilters` contains three IFilter DLL's. These are installed by default, and they are used to index the contents of ZIP files, Text files, and Microsoft Access Databases with the *CollectionConnection* Indexer. These filters conform to the specification of the Microsoft Ifilter interface. Therefore, **after the installation of *CollectionConnection*, the Microsoft Indexing Service will also start indexing the contents of ZIP files, Text files, and Access databases.** This might be unwanted behavior. The filters can easily be uninstalled by executing respectively:

```
uninstallccTxtIFilter.bat
uninstallccZipIFilter.bat
uninstallccMdbIFilter.bat
```

They can be installed again by executing:

```
installccTxtIFilter.bat
installccZipIFilter.bat
installccMdbIFilter.bat
```

Indexer

5 CollectionConnection Indexer

5.1 Introduction

The *CollectionConnection* Indexer is a data mapping tool. It takes source data from, for example, an ADO database connection, an Adlib export file, or an XML file. This data is mapped to an XML structure and put in a file that can be used for high speed full text searching and browsing.

The Indexer can send the index files automatically to a *CollectionConnection* server. In this way, the Indexer can be scheduled to run periodically so the index that the server uses will always be up to date. Alternatively, the index file can be copied manually to a location where the server can access it.

The settings of the Indexer are stored in a file with the '.indexerprofile' extension.

5.2 Creating a new profile

Creating a new profile consists of three steps. First, a license has to be loaded. This license is provided by the reseller. Second, a data source needs to be chosen. Third, the profile must be given a name. The name of the profile will also be the name of the index that is created.

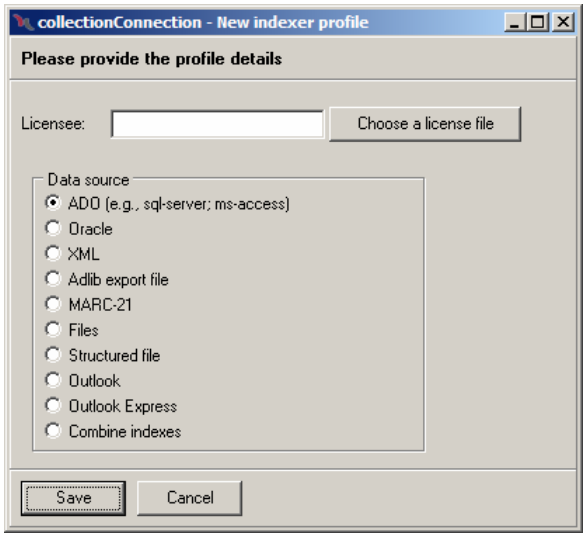


Figure 5.1. New Indexer profile

5.3 Loading and saving a profile

A profile can be given as command line parameter when the program is started, in which case it will open the specified profile. This can be done by double clicking on a profile or by a making a shortcut in Windows, for example with the following command line:

```
D:\program files\collectionconnection\ccIndexer.exe
  "demo.indexerprofile"
```

The following command line parameters can be used:

- `-autostart`. The indexing process is started automatically with the profile. The existing index is overwritten. If this option is specified, the log window is saved to the file '`autostartlog.txt`' in the folder of the Indexer's executable. This file contains error messages might anything have gone wrong.
- `-ftp`. The index is automatically sent with ftp to the server that the profile points to.
- `-mail`. The index is automatically sent to the email address that is specified in the profile.
- `-autoshutdown`. After creating and optionally sending the index, the Indexer is shutdown.

A command line can be also be used in the Microsoft Windows Scheduler (Note that you can not refer to a shortcut in the Scheduler. Rather, you must explicitly refer to the executable and provide the parameter in the Scheduler itself).

If no command line parameter is specified, then the user must manually choose a profile (Figure 5.2).

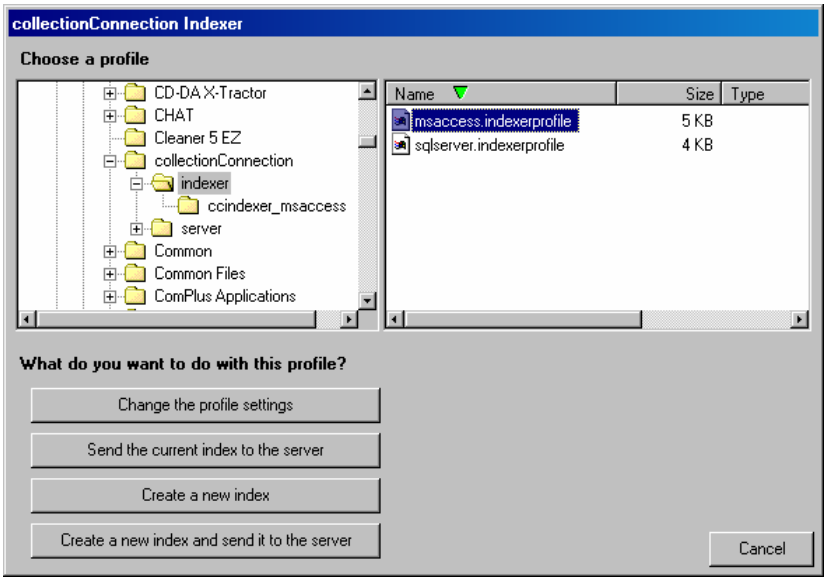


Figure 5.2. Opening an Indexer profile

The following actions are possible:

- *Change the profile settings*. The settings can be changed.

- *Send the current index to the server*. If there is an existing index, it can be sent to the server.
- *Create a new index*. This creates a new index but does not send it to the server. The index can be sent at a later moment or it can be copied manually.
- *Create a new index and send it to the server*. This creates a new index and when it is created, it is sent to the server.

The index file is stored in the directory of the profile.

5.4 The Log window

The log window shows the activities that take place (Figure 5.3). It can be used to monitor the progress of the indexer. Furthermore, it can show the queries that the Indexer sends to the database with the `log queries to screen` setting. This can be used for debugging purposes.

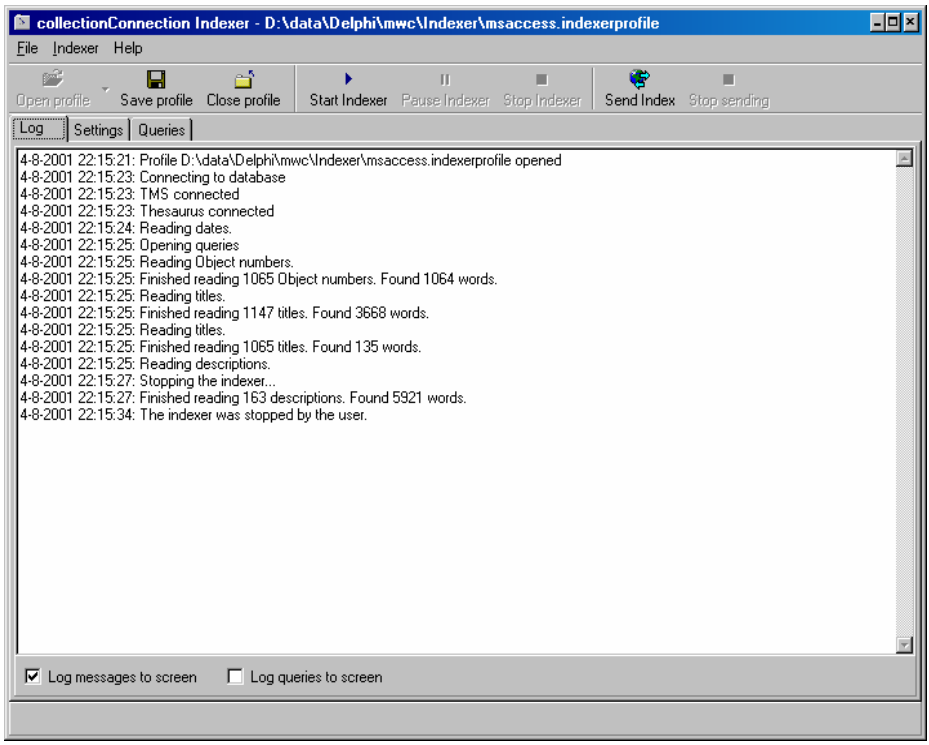

collectionConnection Indexer - D:\data\Delphi\mwc\Indexer\msaccess.indexerprofile
File Indexer Help
Open profile
Save profile
Close profile
Start Indexer
Pause Indexer
Stop Indexer
Send Index
Stop sending
Log
Settings
Queries
4-8-2001 22:15:21: Profile D:\data\Delphi\mwc\Indexer\msaccess.indexerprofile opened
4-8-2001 22:15:23: Connecting to database
4-8-2001 22:15:23: TMS connected
4-8-2001 22:15:23: Thesaurus connected
4-8-2001 22:15:24: Reading dates.
4-8-2001 22:15:25: Opening queries
4-8-2001 22:15:25: Reading Object numbers.
4-8-2001 22:15:25: Finished reading 1065 Object numbers. Found 1064 words.
4-8-2001 22:15:25: Reading titles.
4-8-2001 22:15:25: Finished reading 1147 titles. Found 3668 words.
4-8-2001 22:15:25: Reading titles.
4-8-2001 22:15:25: Finished reading 1065 titles. Found 135 words.
4-8-2001 22:15:25: Reading descriptions.
4-8-2001 22:15:27: Stopping the indexer...
4-8-2001 22:15:27: Finished reading 163 descriptions. Found 5921 words.
4-8-2001 22:15:34: The indexer was stopped by the user.
Log messages to screen
Log queries to screen

Figure 5.3. Indexer log window

If an error occurs during the creation of the index, a detailed error message will be shown in the log file. The contents of the error message can be used by the reseller to determine the cause of the error.

5.5 The Settings window

The settings specify to what server the index must be sent (Figure 5.4).

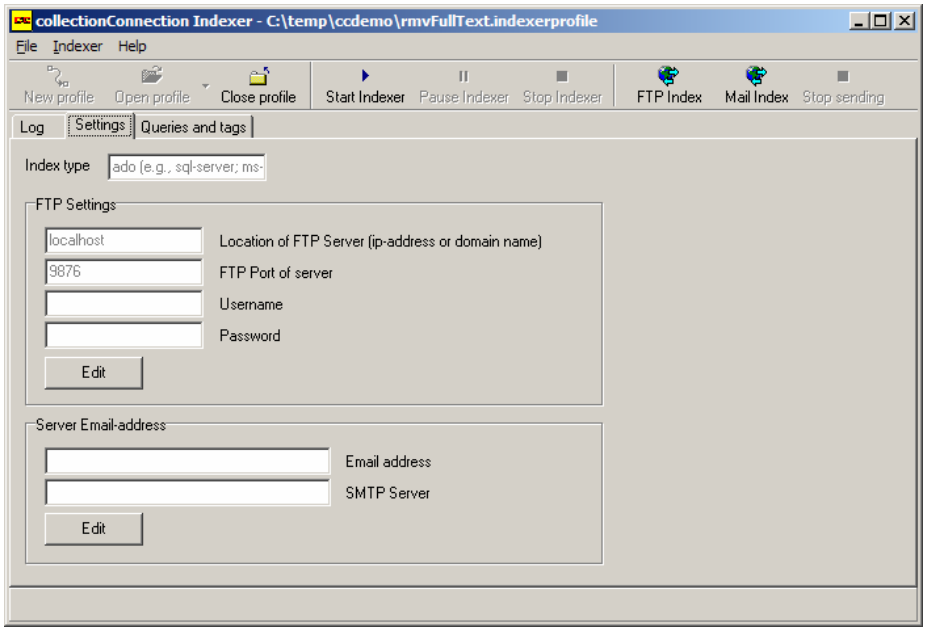


Figure 5.4. The Indexer settings

There are two ways to send the index file to the server. The index can be sent using FTP, and it can be sent using E-mail. Both can also be done with a dedicated FTP or E-mail client. Note that sending large index files by E-mail over the Internet can fail because most E-mail servers use a maximum size of E-mails (e.g., 5MB or 10MB).

5.6 Data source elements

5.6.1 Introduction

The indexer is a mapping tool. Source data must be mapped to the *CollectionConnection* index. Table 5.1 contains for the different kinds of source data the elements that are mapped to the *CollectionConnection* XML tags:

Data source	Source elements
Ado	Fields in queries
Oracle	Fields in queries
XML	Tags
Adlib	Rows in export file
Files	Contents of the file
Structured file	Rows or records in a file
Outlook	Email messages
Outlook Express	Email messages

Table 5.1. Data sources and elements

The remainder of this section explains for each of the data sources how the mapping can be specified.

5.6.2 Scripting

As described in chapter 3, a *CollectionConnection* index consists of XML-records, where each record consists of a number of XML-tags. In order to be as flexible as possible during the creation of the index, *CollectionConnection* includes a scripting language. Next to mapping data from a database or other kind of information source to the *CollectionConnection* index, the data can be manipulated before it is put in the index. There are multiple stages at which scripts can be used:

1. **Global scripting variables**. These variables are persistent between execution of the scripts.
2. **Procedures and functions**. Procedures and functions that are specified here are available to all other scripts.
3. The **index preprocessing** script. The preprocessing script is called before any data is retrieved from the information source. This script can be used to initialize global scripting variables that can be used in the other scripts, for example, creating COM-objects or database connections.
4. The **record filter script**. This script can be used to disable inclusion of a record in the index. The script will be executed after the record has been created. For most index types, both the source data and the mapped record are available in the script. See section 5.6.11.1 on page 66 for the available source data. For Ado/Oracle indexes, the record filter script can not access the underlying source datasets, since there can be more than one and the datasets are only active during the mapping phase. This can be circumvented by adding tags on which the inclusion decision is based to the XML record and basing the decision on that. The variable `ccXMLRecord` contains the mapped XML record. Stating `ccIncludeRecord=True` in the script will include the record in the index. Stating `ccIncludeRecord=False` in the script will disable inclusion.
5. The **tag** scripts. A tag can be created with a script instead of getting the information directly from the data source. The script can use the information that the information source provides. For example, if the source is an ADO database, a script can append two fields from a query to one tag. For a more detailed description, see section 5.6.11.1 on page 66.
6. The **record postprocessing** script. After the XML-record has been created, it can be altered with a script. Tags or attributes can be added or deleted (using the `ccXMLRecord` variable) on the basis of the contents of the script. For example, the script can check whether an image of which the location is specified in a tag actually exists. If not, it can delete the tag. Note that tags that are not specified in the indexer profile itself can be added to the XML-record, but *CollectionConnection* has no knowledge of these tags. This means that these tags will be part of the output of a query, but they cannot be used to search the index or sort the result list.
7. The **index postprocessing** script. Global variables that have been initialized in the preprocessing script can be finalized in the postprocessing script. For example, if a database connection was opened it should also be closed.

Figure 5.5 shows the `queries and tags` window, that shows in the left pane the scripts that can be edited, and in the right pane the chosen script. Pressing the `Edit script` button will open a window in which the script can be edited (Figure 5.6).

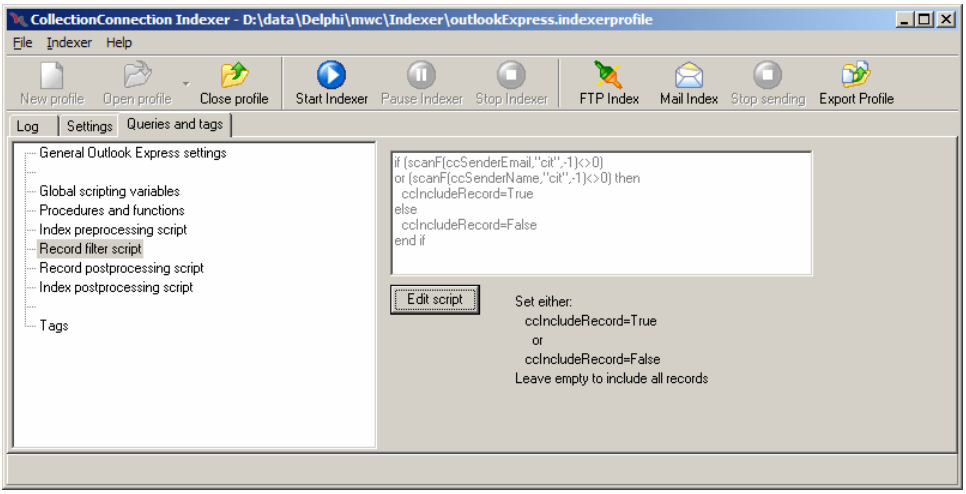


Figure 5.5. CollectionConnection scripts

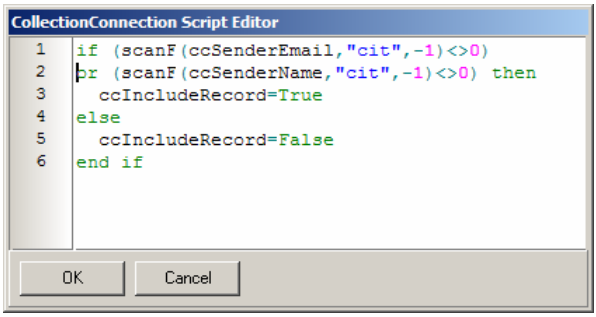


Figure 5.6. The CollectionConnection script editor

An example of the use of scripts is to add all records that are created to a Microsoft Word file. A COM-object of Microsoft Word can be created in the index preprocessing script. In the record postprocessing script, the XML-record is added to the Word file, and in the index postprocessing script, the OLE-object is freed. This could be implemented as follows:

```
Global variables
dim wordapp
dim worddoc
dim wordrange

Index preprocessing script
wordapp = CreateOleObject("Word.Application")
wordapp.Visible = False
wordapp.DisplayAlerts = 0
wordapp.Options.CheckSpellingAsYouType = False
wordapp.Options.CheckGrammarAsYouType = False
worddoc = wordapp.documents.add
```

```
worddoc.select
wordrange = worddoc.Range
wordrange.insertafter("Collection
   Connection"+chr(13)+chr(10))

Record postprocessing script
dim xmld
xmld = Create0le0bject("microsoft.xmldom")
xmld.loadxml(ccXMLRecord)
wordrange.insertafter(xmld.documentElement.selectsing
   lenode("identifier").text+chr(13)+chr(10))
wordrange.insertafter(xmld.documentElement.selectsing
   lenode("title").text+chr(13)+chr(10))
wordrange.insertafter(chr(13)+chr(10))
xmld = nothing

Index postprocessing script
worddoc.saveas("c:\temp\result.doc")
worddoc.close
wordapp.quit
```

The scripting language is Basic. The scripting language can create objects that conform to the COM (common object model) specification. This exposes much functionality, for example for string manipulations, database transactions, internet connectivity, etc. Appendix A contains a reference of the language and an overview of the available functions.

5.6.3 Combine indexes

CollectionConnection indexes can be combined. This means that data from several sources (e.g., objects from a database, books from a text file, author biographies from an XML file) can be put in one *CollectionConnection* index. Figure 5.7 shows the settings for this index type. Each index that is included must be given a name. This name is added to all XML-records as the tag `<ccIndexName>`. This can be useful when searches are being performed on a combined index but not all included indexes need to be searched.

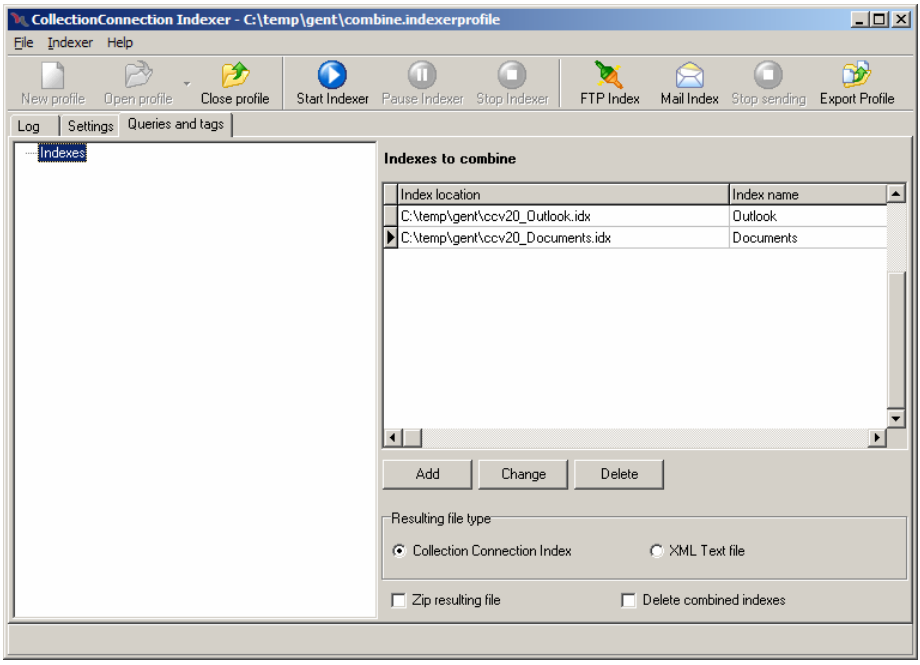


Figure 5.7. The Indexer - combine indexes window

All data from the individual indexes is preserved with the following exceptions:

1. Only the thesaurus (see section 5.6.5.4 on page 48) of the first index in the list is included. This is to prevent thesaurus conflicts. Assignments of objects to thesaurus nodes are preserved as long as a node is available in the included thesaurus.
2. Dynamic content links are not preserved.

There are two kinds of outputs. First, the resulting index can be a *CollectionConnection* Index itself. Second, the index can be an XML text file. Optionally, the resulting file can be compressed.

Indexes can be deleted after they have been put in a combined index. This option can be used when the individual indexes will not be used themselves. It is especially useful in combination with batch processing of indexer profiles (see section 5.6.4).

5.6.4 Batch process index creation

CollectionConnection can process multiple indexer profiles in one batch. Figure 5.8 shows the screen in which the indexer profiles to be processed are specified. The indexes are created in the order as listed. Batch processing is especially useful to create a number of indexes (for example, an index with Outlook email messages and an index with all MS-Word documents) that are in the last stage combined to one index.

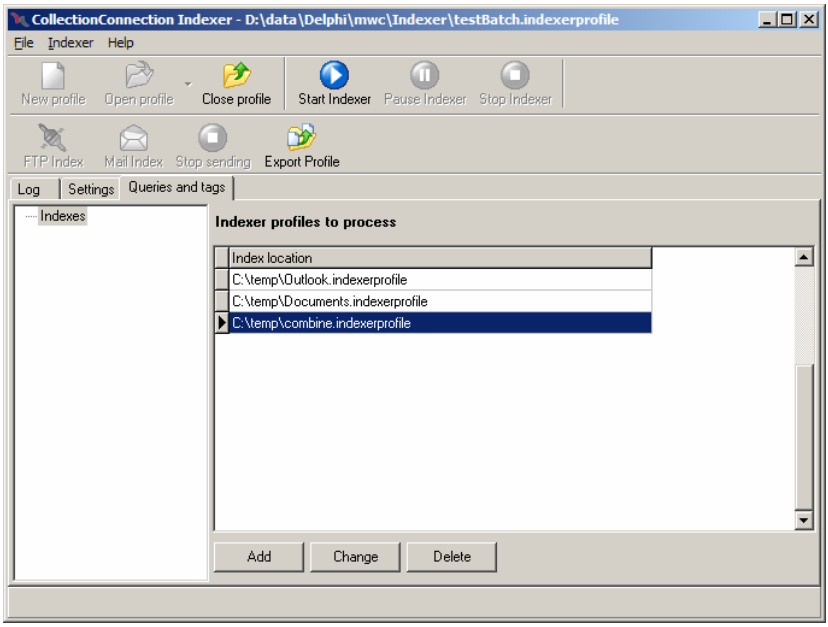


Figure 5.8. Batch processing of indexer profiles

5.6.5 ADO/Oracle specific settings

5.6.5.1 ADO Database connection

To get information from an ADO compliant database, a valid database connection needs to be specified. This can be done manually (Figure 5.9), but the generic Windows dialogs to specify a database connection can also be used by pressing the button with the three dots shown on the right side of Figure 5.9. Figure 5.10 and Figure 5.11 show the first two dialogs that are presented. Further dialogs depend on the chosen data provider.

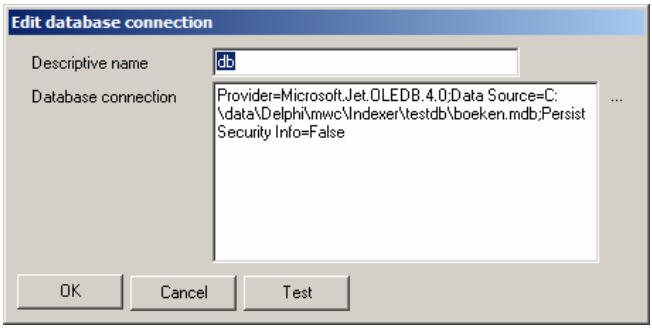


Figure 5.9. Manually specifying a database connection

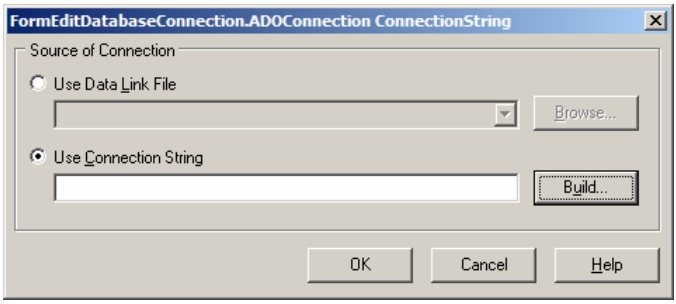


Figure 5.10. Generic windows ADO database connection dialog

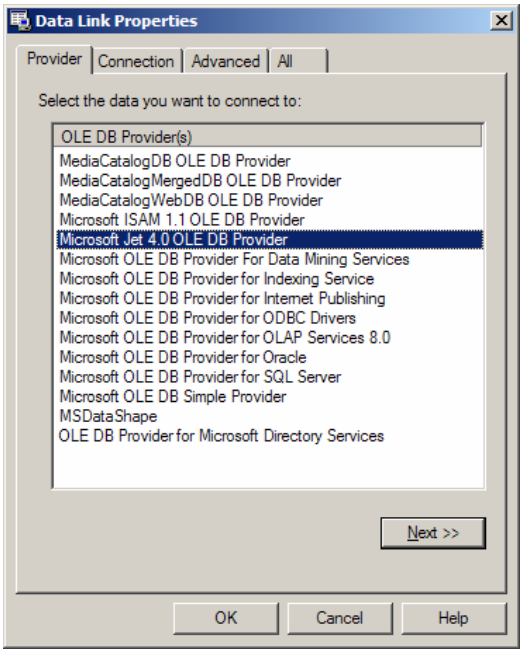

Data Link Properties
Provider
Connection
Advanced
All
Select the data you want to connect to:
OLE DB Provider(s)
MediaCatalogDB OLE DB Provider
MediaCatalogMergedDB OLE DB Provider
MediaCatalogWebDB OLE DB Provider
Microsoft ISAM 1.1 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft OLE DB Provider For Data Mining Services
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for Internet Publishing
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
MSDataShape
OLE DB Provider for Microsoft Directory Services
Next >>
OK
Cancel
Help

Figure 5.11. Generic windows ADO database connection dialog with ADO compliant data providers

5.6.5.2 Oracle Database connection

The *CollectionConnection* Indexer can connect to Oracle directly without using ODBC or ADO. To establish the connection, the username, password, server, port, and sid need to be specified (see Figure 5.12).

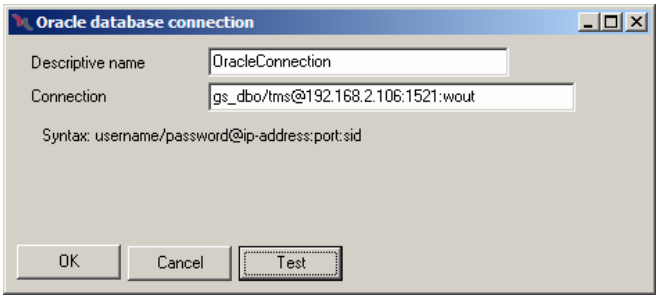


Figure 5.12. Oracle Database connection

5.6.5.3 Database structures

A database structure contains the relations between tables in the database. Not all tables from the database have to be added to the structure, which increases the clarity of the indexer profile. The database structure is edited with the Data Model Editor. It contains information about tables, links between them, some entities of the data domain named "Fields" (not the same as fields in database tables although some entities can be totally equal to fields in database). In addition this description includes list of operators used in conditions (such as "is equal to" or "is more than") and list of custom (manually defined) conditions which can be used in queries.

Though being easy to use and self-explanatory, structure editing requires some understanding of the relational databases handling. Users totally unfamiliar with database management should better leave this dialog for an database operator or another experienced user. This dialog contains the following pages:

- Tables - describing tables taking part in the data model
- Fields - describing the entities which end-user can operate with to build queries
- Operators - describing operations (like comparisons) upon the data model's fields
- Conditions - describing custom (user-defined) conditions which then can be used in queries

Each of the pages will be described in detail.

Use the Load and Save speed buttons to load or save the data model from/to an INI-like textual file.

Table page

Use this page to choose the tables taking part in the data model (Figure 5.13). This page contains two main panels: tables with links between them and properties of the current chosen (active) table. Tables are represented by movable windows with fields list inside. If the Data Model Editor is not currently connected to the database then the tables contain just the `<not connected>` caption. The active table is indicated by color and wider border. Links between the tables are represented by lines between the tables. The active link is thicker then others. To select a table or link just click on it. Properties of the selected table are shown in the right panel. To edit properties of the current link, just double click on it and the Edit Link dialog will appear.

To add or delete tables and links use the corresponding buttons at the top of the dialog (they should appear when "Tables" page is selected) or right-click to get a popup menu. The table properties are:

- *Table alias* is an alias for the table in generated SQL statements (optional but needed if you link the table more than once on different conditions)

- *Table hint(s)* is intended to specify locking method used in MS SQL Server syntax (like `NOLOCK`, `ROWLOCK` or `READCOMMITED`)
- *Quote table name* checkbox means if the table name should be put in double quotes in SQL statements; useful for table names including spaces and national characters.

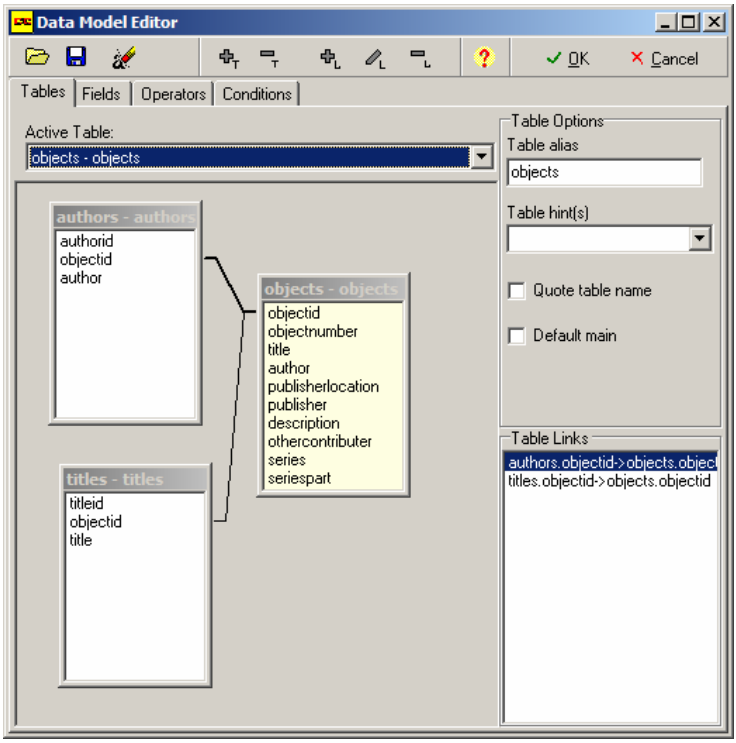


Figure 5.13. Creating a database structure

NOTE: any particular database table can be added to structure more than several times with different aliases. It is necessary to eliminate the ambiguities when two tables have several link paths from one to another.

Editing links between tables.
The two listboxes in Figure 5.14 ("Table1 fields" and "Table2 fields") display fields of main table (the edited one) and linked table. The third listbox (between them) allows to select the operator (equal by default) used in the join condition. The user should select the fields and the operator used in join condition and click Add. The "Quote field names" checkbox means if the field names should be put in double quotes in SQL JOIN conditions; useful for field names including spaces and national characters. The "Joined fields" listbox displays the fields that are in the join condition. The delete button deletes a selected condition. The clear button clears all conditions completely. Join type group of radio buttons allows you to specify the type of join (inner, left outer, right outer, or full join).

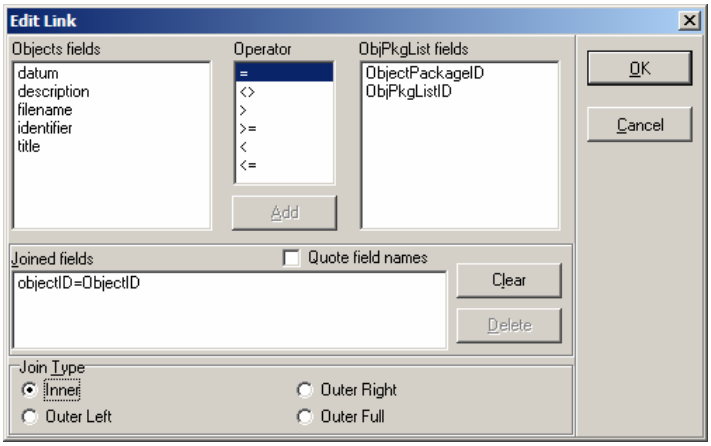


Figure 5.14. Editing a link between tables

Fields editor
Figure 5.15 shows the fields editor. There are two main types of fields:

- Data Field: a field which corresponds to a particular field in a table.
- Virtual Field: a calculated field which is defined by some expression containing several fields, operators (`+`, `-`, `||`, etc.), constants, and even function or stored procedure calls.

The fields tree shows the list of all defined fields. To add a data field, right-click the list and choose Add Field. Then, in the appearing dialog, choose the table, its field and click the OK button. To add a virtual (i.e., calculated) field, right-click the list and choose Add Virtual Field. To delete a field, choose it from the list, right-click it and choose Delete Field. All operations above can be done also trough corresponding speed buttons (at the top of the dialog).

To edit a field, choose it from the list and change its properties in the editor appearing in the dialog's right part. To move the field from one group to another (or to change its appearance order within the group), drag it to the appropriate place. The field's property editor has General, Operators and Value Editor pages described below.

General page. The display name is a string meaning how the field name will be displayed for an end-user. The group name means the group holding the field. This "group" doesn't affect any built SQL text; it's introduced just for the user's convenience when it comes to choosing a field in the visual query builder. Choose a group from the drop-down list or enter a new group name. The following parameters are different for data and virtual fields. For a data field: the “Quote field name” checkbox means whether the field name should be put in double quotes in SQL statements; useful for field names including spaces and national characters. For a virtual field: “Field type” is the data type for resulting field. See Field types for details. “Expression” is a SQL expression for calculating the field's value.

The Operators page shows a list of all operations applicable to the field. Add or remove operations by corresponding buttons. Use the Clear button to remove all operators from the list.

The Value editor page defines how the field value (or, rather the value of a parameter to which the field is compared and having the same type as the field) will be edited in the visual query builder. Choose way of editing form Editor type drop-down list. Editor parameters vary depending on editor type:

- Auto: means that the most appropriate value editor will be used depending on the field's data type and the operator which is used in the condition. For example: in a date field a DataTime Picker control will be shown; and for a boolean field: the user will get an ability to select the value from the list of two items: False and True.
- Edit: the field will edited in a plain text edit field. "Default value" is the value for the field if one isn't specified. The "Use mask" checkbox indicates if an editor mask should be used when editing the field.
- List: the user will be prompted for field value by a fixed list. The "Items column" stands for values accepted as the choice result (and actually inserted into SQL text); Values are captions displayed to user in the pop-up list.
- Custom: This option is currently not used in *CollectionConnection*.
- SQL: Similar to the "List" value editor type, but the list of available values is the result of some SQL query to the database. The first column of the resulting data set is interpreted as the actual value which will be used in the resulting SQL statement and the second one is used as the text shown to user.

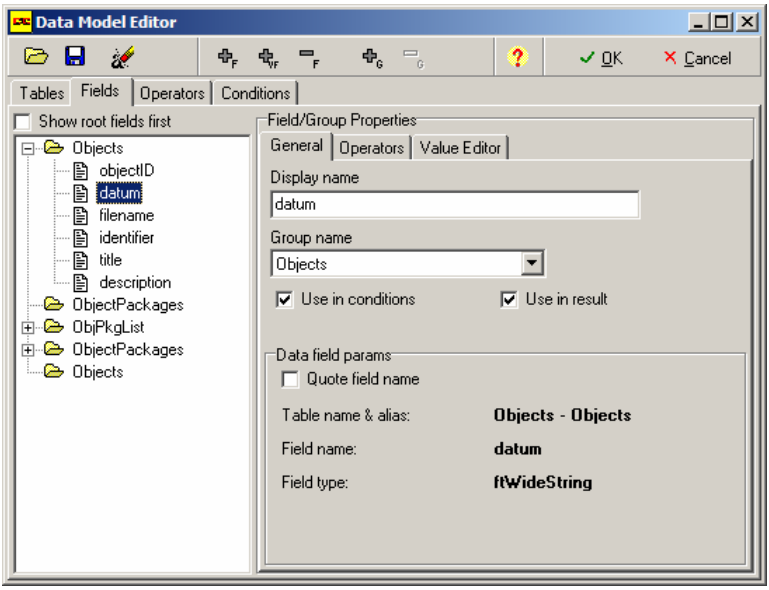


Figure 5.15. Editing fields

Note that for BLOBS, the setting 'use in result' is not checked by default. To include BLOB fields in a query in order to attach them to an index, this setting must be enabled manually.

Operators editor
The operators editor (see Figure 5.16) defines the operators which can be used in conditions (such as `equal to`, `less than`, and others). The list in the left part shows the defined operations. Add or delete one by right-clicking the list and choosing the appropriate topic from the popup menu or using corresponding speedbuttons on the top of the page. To edit an operation choose it from the list and modify its properties in the right part of the page. The operator name is the internal identifier for the operator. The display name specifies how the operation will be displayed the user working with visual query editor.

The SQL expression is a template for expressing the operator in the generated SQL query. It may contain any correct SQL expressions (operators such as `=`, `>`,`<`, functions or even names of stored procedures) and the following special variables:

- `@f` - is substituted with the field's name
- `@1` and `@2` - are substituted with 1st and 2nd constant parameter.

Constant value variables must be enclosed by single quotes in the template expression (the quotes will be deleted for expressions where they are not necessary, for example for integer or real data types). Some examples are:

- For the simple "is equal to" operator the format string is: `@f='@1'`
- The "starts with" operator has the following format: `@f LIKE '@1%'`
- To provide a case insensitive "starts with" operator use the following format string: `LOWER(@f) LIKE LOWER('@1%')`

"Values format" is a string which will separate parameters (in case there are two of them) in the visual query editor. For example for the "between" operator, the value format field contains 'and', which means that this word will be placed between two constants values in condition: Some field is between 1000 and 2000.

"Type of expression" means the expected type for a parameter. It can be the field type itself or some other type (see Field types).

Value kind defines kind of data which will be used in condition:

- "Scalar" means that a single value is needed: one string, one number, etc.
- The "List" type requires a list of scalar values separated by comma. For example, having this option checked, when the user enters a, b, c as parameter values, it's treated as 'a', 'b', 'c' instead of 'a, b, c' in the generated SQL text.
- The "Sub Query" type means that the operator requires an SQL SELECT statement as value in the right part of the condition. To build this statement the query panel opens a separate dialog.

“Apply to types” is a list of checkboxes defining field types to which the operation is applicable.

The operators page also has two additions buttons: "Add operator to appropriate fields" and "Add/Update default operators":

- The "Add operator to appropriate fields" button can be used to add a newly created operator to all fields in the structure that have the appropriate type.
- The "Add/Update default operators" button adds the operator to the list with default operators or updates an existing default operator.

CollectionConnection has the following predefined operators (Table 5.2):

Name	Display name	SQL expression
Equal	is equal to	@f = '@1'
NotEqual	is not equal to	@f <> '@1'
LessThan	is less than	@f < '@1'
LessOrEqual	is less than or equal to	@f <= '@1'
GreaterThan	greater than	@f >'@1'
GreaterOrEqual	greater than or equal to	@f >= '@1'
IsNull	is null	@f is null
InList	is in list	@f in (@1)
StartsWith	starts with	@f like '@1%'
NotStartsWith	does not start with	not(@f like '@1%')
Contains	contains	@f like '%@1%'
NotContains	does not contain	not(@f like '%@1%')
Between	is between	@f between '@1' and '@2'
InSubQuery	InSubQuery	@f in (@1)

Table 5.2. Default operators

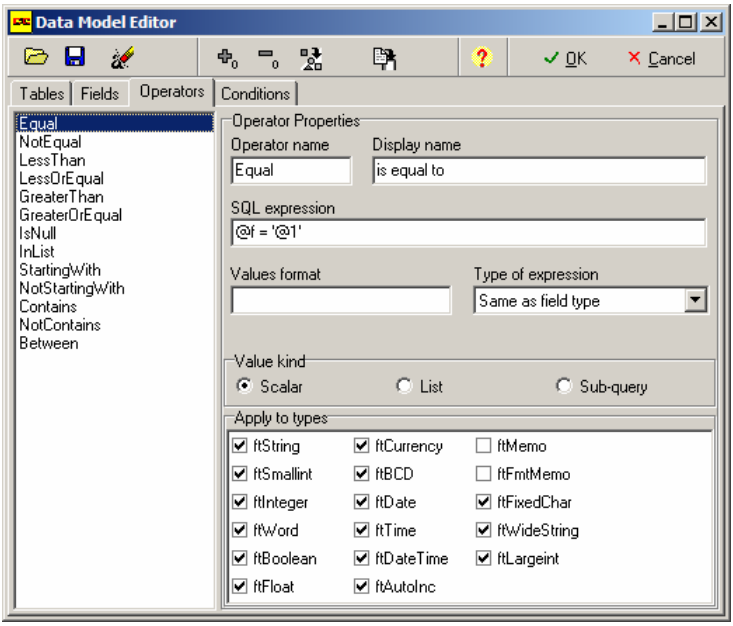


Figure 5.16. Editing operators

Conditions editor

The conditions editor (Figure 5.17) allows to specify custom conditions which then can be added into the query. Use this ability to declare the conditions which are used very often or which can not be defined using a standard `<field> <operator> <value>` scheme. For example, the following query:

"select all orders which are made on the same day as when the new employee was hired"

does not match the standard condition scheme used in the query panel but can be simply defined in SQL:

```
Orders.SaleDate = Employee.HireDate
```

To provide our users with an ability to build such queries we should define a new condition with the following properties:

- DisplayFormat: "Order made when new employer was hired"
- SQL Expression: "`Orders.SaleDate = Employee.HireDate`"

Here `DisplayFormat` is the text which will be shown for the users and which will be added into the query panel as a condition. The DisplayFormat property can contain fields marked by the `@` symbol. It means that the user will need to enter some value in this place. Each `@` symbol correspond to a variable which should be defined in the SQL Expression property. Variables are specified by the `@` symbol and follow numbering starting from 1. The `@1` variable corresponds to the first `@` symbol in the `DisplayFormat` property, `@2` to the second, etc.

For example, the following condition "Address contains <some value>" can be defined by standard methods (using `<field> <operator> <value>` scheme) but can be also declared as a custom condition with the following properties:

- DisplayFormat: "`Address contains @`"
- SQL Expression: "`Customer.Addr1 LIKE '%@1%'`"
- Value Data type: `String`
- Value kind : `Scalar`

The last two properties can be specified only when you define one or more values in the DisplayFormat property. In such cases new item(s) will appear in "Values" list box (at the bottom of the "Conditions" page). For each value its data type (String, Integer, Date and so on) and its kind (Scalar, List or Sub-Query) can be defined. See the Operators editor topic for more information about kinds of values.

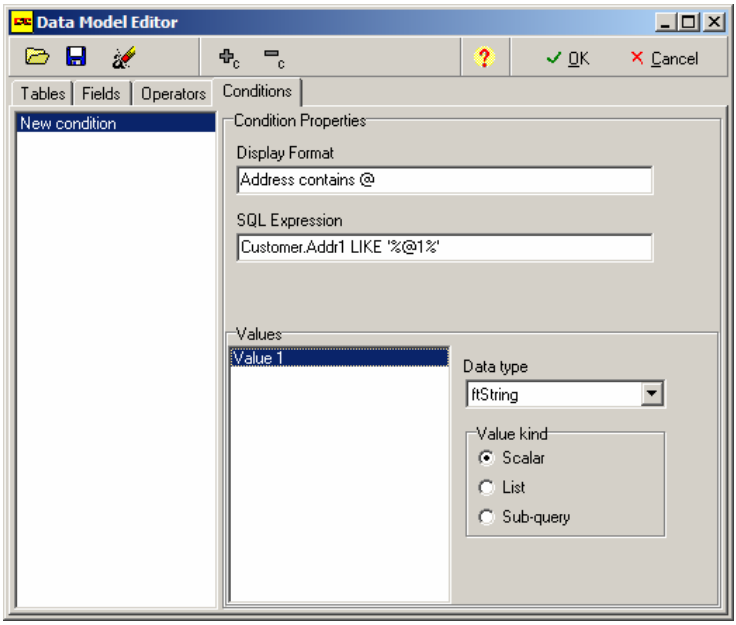


Figure 5.17. Editing conditions

5.6.5.4 Queries

Queries provide the data for the following *CollectionConnection* information elements (see also chapter 3):

- Tags
- Dynamic Content Links (DCL)
- List with records
- List with thesaurus nodes
- Thesaurus structure
- Thesaurus terms
- Collection list
- Collections

A query can be used for multiple tags or dynamic content links. For that reason, there is a separate pane in the Indexer window named 'queries'. These queries are the basis for tags and dynamic content links. The information elements each have their own query.

Most of the queries have requisite fields. Table 5.3 describes the fields that are required per query:

Query	Required fields	Explanation
Queries used for tags	ccObjectID	XML records (also called objects) can get their information from multiple 1:n queries. Such queries must be linked and CollectionConnection must know how they are linked. For that reason, each query must have a field ccObjectID of the datatype Integer that identifies it's objects.
List with object identifiers	ccObjectID	This query is included to specify what objects must be indexed. Although this could in principle also be done via the tag-queries themselves using a 'where' clause or by using the record filter script, the separation proves more versatile and easier to adapt.
List with thesaurus identifiers	ccNodeID	Each node in the thesaurus must be identified by an integer field. This query specifies what nodes must be included in the Index.
Objects in thesaurus	ccNodeID ccNodeName ccObjectID ccTerm	This query links the objects to the thesaurus nodes. The node name, node ID, and term are used to add a reference to the thesaurus node, to the object's XML record. The node name is the hierarchical node notation, the term is the thesaurus term.
Thesaurus nodes and terms	ccNodeID ccNodeName ccTerm ccLanguageID ccTermTypeID	This query specifies the structure of the thesaurus. The query must order the nodes alphabetically on ccNodeName. A node can have multiple terms, each with it's own language and term type. ccLanguageID and ccTermTypeID are Integer fields.
Collection list	ccCollectionID ccCollectionGroup ccCollectionName ccCollectionDescription	Objects can be grouped in collections. Each collection needs an Integer field ccCollectionID to identify itself. Collections can be grouped themselves using the ccCollectionGroup string field. Each collection can also have a name and a description. (If a collection description is not available in the structure of the database, specify `"" as ccCollectionDescription` in the field part of the query.
Objects in collection	ccObjectID %s	This query specifies what objects are in a collection. Note that %s is not a field but a parameter. During indexing, %s is

	replaced by the appropriate ccCollectionID. (for example: select objectID as ccObjectID from ObjPkgList where ObjectPackageID=%s order by orderpos)

Table 5.3. Required fields per query

All queries use the same query editor (Figure 5.18). A query can either be created manually or with the visual query builder. In the result field editor of the visual query builder (see Figure 5.21 on page 53), field names from the database can be mapped to the required field names as stated in Table 5.3. Queries that are created with the query builder can be altered manually. Queries that are created or altered manually can not be edited with the query builder.

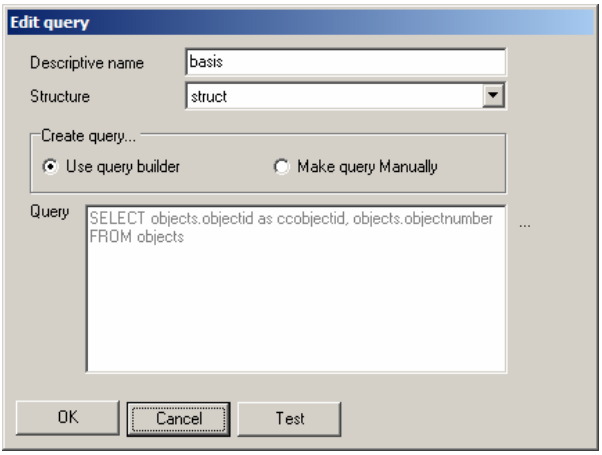


Figure 5.18. Choosing to edit a query manually or with the query builder

The query panel (Figure 5.19 and Figure 5.20) is the main interface element for creating queries. Each row in the query panel represents one condition or title of the conditions group (bracket). Each condition group has a linking type parameter which specifies how conditions in group are connected to each other. The conditions and groups of conditions can be organized in a hierarchy with an unlimited number of levels. The root of this hierarchy is the "root group" (or "root bracket") which contains all conditions and groups of conditions of the first level.

There are two types of conditions: simple conditions and custom conditions. Simple conditions consists of three main parts: a field (one of the fields defined in data model), an operator (such as "is equal to" or "less than"), and the values. The values list can be empty as in case of "is NULL" operator. Custom conditions are defined in the data model and represented by some textual description and (possibly) one or more values. Each row also contains an "enable/disable" check box and a special "row button" which can be used to delete the current condition or to add a new condition.

The last row in query panel is the "addition button" which can be used to add one simple condition (left click) or to perform the other task available through the context-menu (right-click): add bracket, and add custom condition.

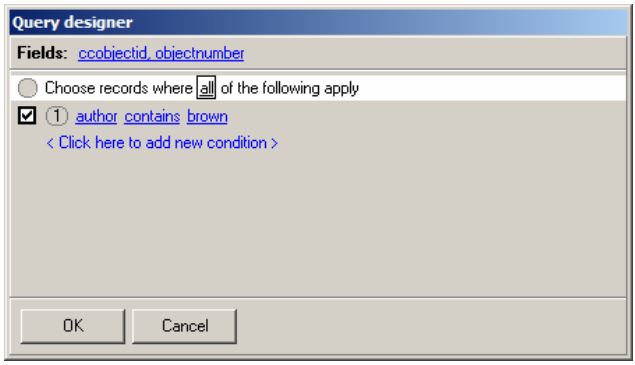


Figure 5.19. Creating the query

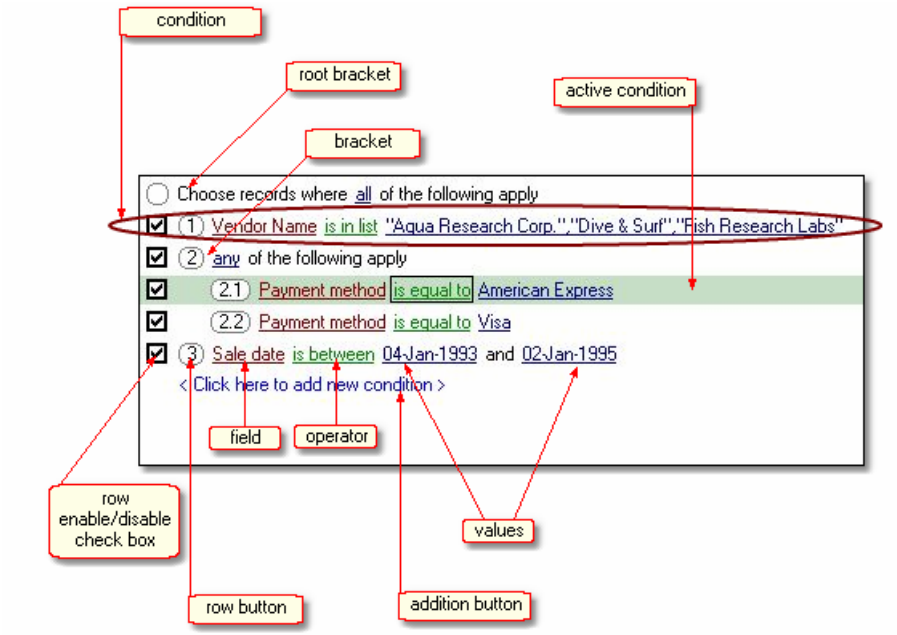

condition
root bracket
active condition
bracket
Choose records where all of the following apply
1 Vendor Name is in list "Aqua Research Corp.","Dive & Surf","Fish Research Labs"
2 any of the following apply
2.1 Payment method is equal to American Express
2.2 Payment method is equal to Visa
3 Sale date is between 04-Jan-1993 and 02-Jan-1995
< Click here to add new condition >
field
operator
row enable/disable check box
values
row button
addition button

Figure 5.20. The query panel

The following keyboard shortcuts can be used:

- Ins: Adds a new simple condition after the current condition.
- Alt+Ins: Brings up context menu. Using this menu you can add new custom condition or bracket
- Ctrl+Del: Deletes the current row or bracket.

- Ctrl+<up arrow>: Moves the current row up in query panel. In other words: it changes the places of the current row and the row above it on the same level.
- Ctrl+<down arrow>: Moves the current row down in the the query panel. In other words: it changes the places of the current row and the row below it on the same level.
- Ctrl+<left arrow>: If the current condition belongs to some group then it is moved one level up.
- Ctrl+<right arrow>: If the previous row of the same level is a group, then the current condition will be added to that group.

Result fields
In the query designer, the fields for the query can be selected by clicking on the fields hyperlink. The result field editor (Figure 5.21) shows the fields that are available in the data model. Fields are grouped as defined in the Structure Editor. Unclassified fields (i.e., not belonging to any group) are represented as the tree's root level. To add a field select it in the list and press the add button. From the appearing menu select how the field should be added. Variants are:

- `as is`: the field is added as is; no formulas are applied.
- `MIN, MAX, AVERAGE, COUNT, SUM`: the field is included as an argument of the corresponding SQL aggregate function.

To create a calculated field, follow the next steps:

- Choose a field from the tree and add it to the list
- Choose a second field from the tree; select the field you just added in the Result fields list
- Press the add button and choose the appropriate operation from the drop down menu (Add to result field, Substract from result field, Multiply by result field, Divide by result field)
- Using the previous two steps, you can add more fields to the expression, e.g., `field1+field2*field3-field4`.
- To insert brackets, select the result field and click the Expression cell. In the edited expression, fields are shown as `@1, @2,...,@n`.

To remove a field select it in the Result field list, press right mouse button and choose the "Delete result field" menu item.

The result fields editor also allows to specify a field to sort on. This option may not be used in the *CollectionConnection* Indexer!

To add the field to the sorting order, select it in the Result field list, press the right mouse button and choose "Sort result field" menu item. Then in the sorted fields list below use can change type of sorting by clicking the right mouse button.

You can change name of the result field: just select it and then rename it. This might be necessary for queries that require specific field names present (see also Table 5.3 on page 50).

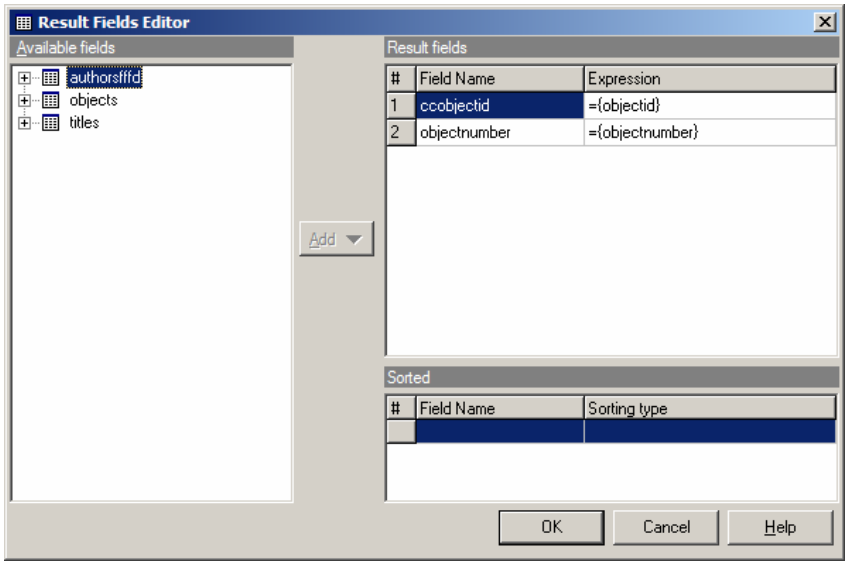


Figure 5.21. Result fields editor

5.6.5.5 XML specific settings

There are two generic settings if the source is an XML file. First, the filename needs to be specified. Second, the tag that signifies a record in *CollectionConnection* must be specified. Tags within this record-tag can become tags in the *CollectionConnection* index.

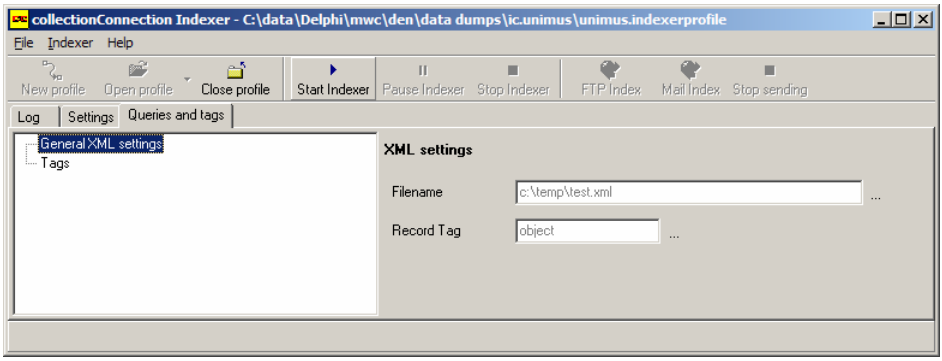


Figure 5.22. Generic XML settings

5.6.6 Adlib specific settings

There are two generic settings if the source is an Adlib file. First, the filename needs to be specified. Second, the 'end-of-record' marker in the Adlib file must be specified. The text between two such markers contains the data for one *CollectionConnection* XML-record.

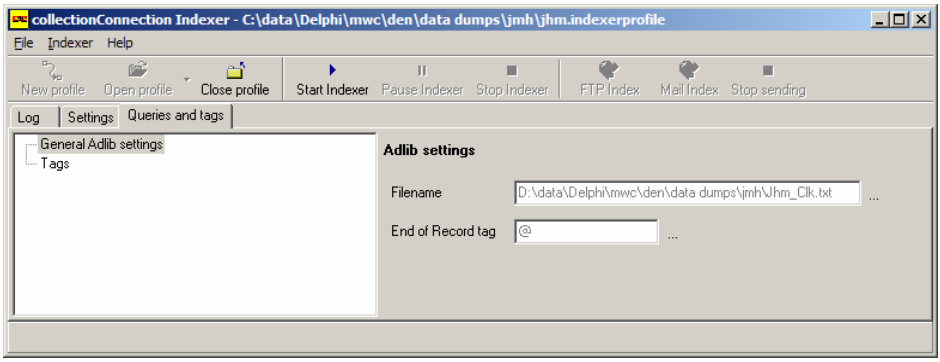


Figure 5.23. Generic Adlib settings

5.6.7 Files specific settings

The indexer can be used to index a number of files. In this kind of index, each file will become a record. The files to be indexed are specified in two stages: specifying the folders to search in, and specifying matching patterns to determine what files in the specified folders need to be indexed and how they should be indexed (see Figure 5.24).

⚠ For indexing large files (for example large MS-Access databases or large Zip archives), the indexer needs enough internal memory. Especially in the case of Zip archives, the amount of memory can exceed the size of the Zip files multiple times. Note that the size of archived files is not related to the memory needed by the *CollectionConnection* server, except when the original file is requested from the index as a binary attachment or when the complete contents of the file is requested as text.

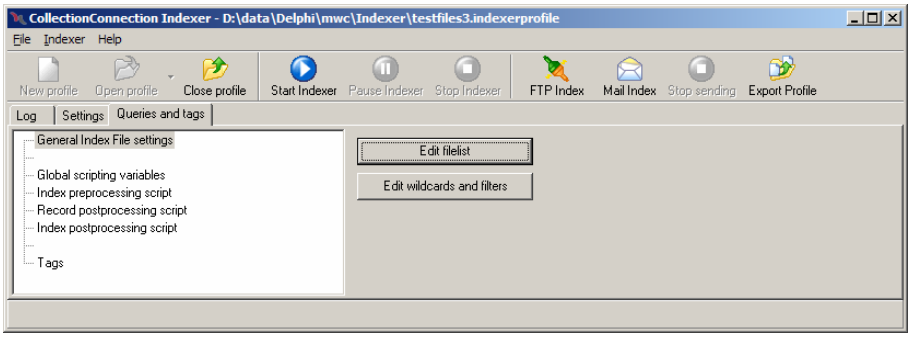


Figure 5.24. Generic file indexing settings

5.6.7.1 Specifying folders

Specifying the folders to search in is done by checking the appropriate items in the tree as shown in Figure 5.25. Checking a folder means that all files in that folder and its subfolders will be matched against the patterns. Checked files and folders can be unchecked individually.

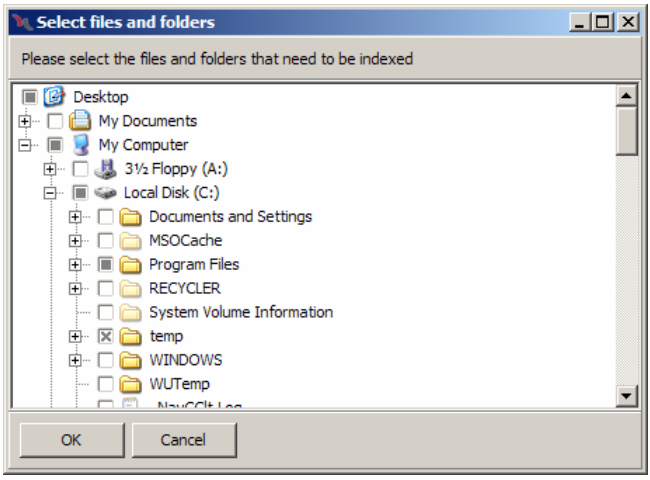


Figure 5.25. Specifying a folder

5.6.7.2 File pattern matching and filter specification

CollectionConnection uses Regular Expressions to specify the files that need to be indexed. Using Regular Expressions is more complex than simple wildcard specifications (such as *.doc), but also much more powerful. Multiple regular expressions can be entered (see Figure 5.26). For each regular expression, a filter needs to be specified (see section 5.6.7.3). Furthermore, the indexer can be restricted to index files that have certain attributes set, that are created or modified before and/or after a certain date, and the minimum and maximum size can be specified (see Figure 5.27).

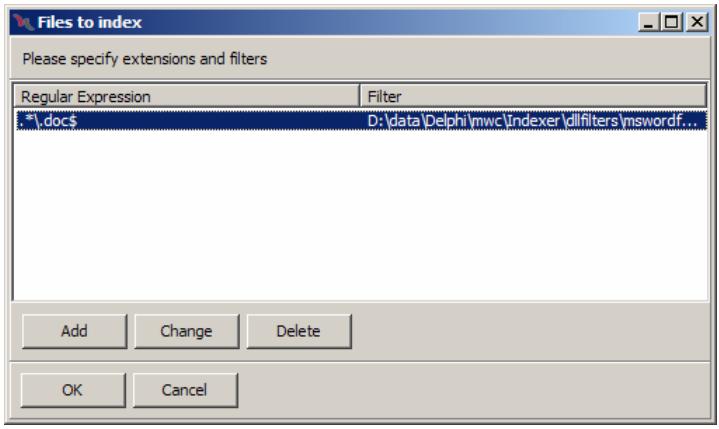


Figure 5.26. Pattern matching and filters - overview

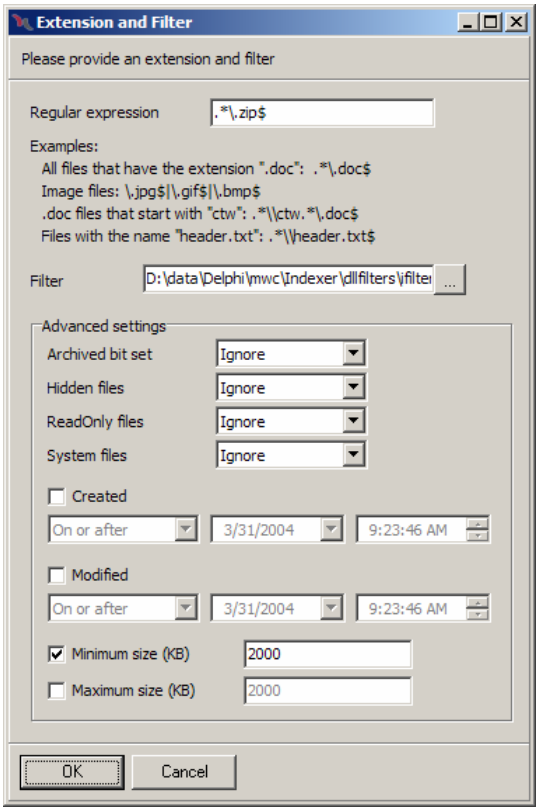


Figure 5.27. Pattern matching and filters - settings

A file can match multiple regular expressions. Then, all accompanying filters will be applied, but the file will correspond to one record in the index.

⚠ Make sure to check all tags that use filters after a the Regular Expression or location of a filter specification has been changed!

5.6.7.3 Text providers for file indexing

CollectionConnection is very flexible with respect to indexing of files. Each file that matches the Regular Expression is put through a filter. This filter will transform the document format into text. A filter can specify the to be indexed content of a file in several fields, e.g., author, date of creation, size, content, etc. Filters are Dynamic Link Libraries with a very simple interface. There are 5 functions in the DLL's:

- *ProcessFile*: `Function(filename: PChar): Boolean; stdcall;`
  On the call of this procedure, the filter should process the file with the given filename. On success, the filter will return 'True'.
- *getFieldListLength*: `Function(fileName: PChar): DWord; stdcall;`
  This procedure will return the size of the fieldlist. This function should be used to determine the size of the buffer to prepare before getFieldList is called.

- *getFieldList*: `Procedure(Buf: PChar; maxlen: LongInt); stdcall;`
  This procedure will return a list with fieldnames in the Buffer. The fieldnames are separated by linefeeds.
- *getFieldLength*: `Function(fieldname: PChar): DWord; stdcall;`
  This procedure returns the length of the field. The procedure should be used to determine the size of the buffer that will be prepared before getField is called.
- *getField*: `Procedure(fieldname: PChar; Buf: PChar; maxlen: LongInt); stdcall;`
  This procedure will put the contents of the field in the buffer.

There are eight filters included with the indexer:

- *MSWordfilter*: A filter for Microsoft Word documents. This filter provides more detail than the Ifilter (see below), because next to the base text, properties such as author, word count, and creation date are specified as fields.
- *HTML filter*: For HTML files.
- *MP3 filter*. For ID3 tags in MP3 files.
- *EXIF filter*. For EXIF tags in JPEG images.
- *Text filter*: For plain text files.
- *EML filter*. This filter will parse files that contain RFC822 formatted messages, for example, from POP3 servers, Outlook Express, or NNTP newsgroup servers. The filter will also extract the text from attachments. Compressed attachments (with zip or rar) will be uncompressed, after which the decompressed files will be put through the Ifilter interface to extract their text.
- *Ifilter*: A proxy for the Microsoft Ifilter interface. This interface is available on Windows 2000, Windows XP, and Windows Server 2003 systems and is usually used by the Microsoft Indexing Server. The Microsoft Indexing Server does not have to be active in order to be able to use Ifilters in *CollectionConnection*. Microsoft Office documents can be indexed with the Ifilter by default, and several third party products are available for other kinds of file types, e.g. Adobe PDF, Microsoft Visio, etc. *CollectionConnection* contains three of such third-party Ifilters: one to index text files, one to index files that are in Zip-archives, and one to index Microsoft MS-Access databases (.mdb files). Note that MS-Access databases are indexed without structure (tables/fields); all text in such a database is seen as one large text string. To be able to search in tables/fields (and thereby make use of the structure in the database), an MS-Access database can also be indexed as an Ado source (see section 5.6.5 on page 39 and onward) or as a structured file (see section 5.6.8 on page 58).
- *Binaryfilter*: This filter will return the file as it is in binary form. This filter can be used to add whole files as attachments to the index, and it therefore provides an excellent way to distribute documents (e.g., Microsoft Word documents, PDF files, images) over the internet. The binary filter can also be used to process files that contain plain text, for example, Microsoft Visual Basic or Borland Delphi source code files.

Custom filters can easily be created due to the simplicity of the interface.

Like with fields of a SQL-query, the fields of a filter must be mapped to tags in the resultset with the tags-settings (see section 5.6.11 on page 64).

5.6.8 Indexing structured files

The *CollectionConnection* Indexer can index single files that have their data in rows and columns. Some examples are Microsoft Excel files, plain text files, tables in HTML files, and tables in Word files (see Table 5.4 for the complete list).

File type	Explanation
MS-Excel	MS-Excel directly without OLE
Text file	Fixed width or field delimiters are both supported
XML file	XML file
HTML file	Table in HTML file
Paradox	Paradox tables (without BDE)
Dbase	dBase tables (without BDE)
Lotus 1-2-3	Lotus 1-2-3 directly without OLE
Quattro pro	Quattro-pro directly without OLE
MS-Access	Single table in MS-Access database using DAO/MS-Jet (MS Access itself is not required).
MS-Word	Table in MS-Word document (MS Word is required)
ADO	Single table from an ADO provider

Table 5.4. Supported structured file types

The Import Wizard is used to specify the way in which *CollectionConnection* must retrieve the data from the file. First, it must be specified what kind of source is being used (Figure 5.28).

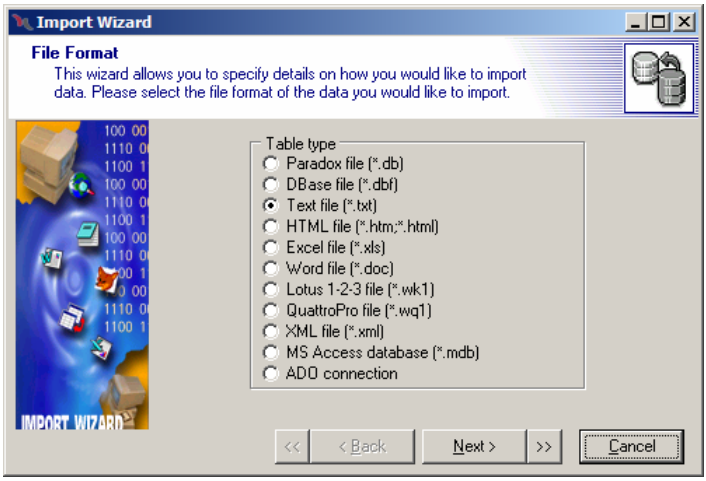

Import Wizard
File Format
This wizard allows you to specify details on how you would like to import data. Please select the file format of the data you would like to import.
Table type
Paradox file (*.db)
DBase file (*.dbf)
Text file (*.txt)
HTML file (*.htm;*.html)
Excel file (*.xls)
Word file (*.doc)
Lotus 1-2-3 file (*.wk1)
QuattroPro file (*.wq1)
XML file (*.xml)
MS Access database (*.mdb)
ADO connection
IMPORT WIZARD
<< < Back Next > >> Cancel

Figure 5.28. Structured file – Choosing a file format

Second, the source file must be specified (Figure 5.29).

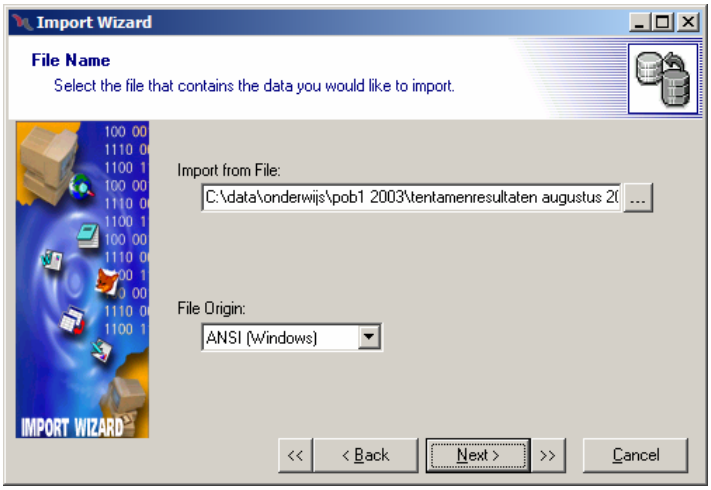


Figure 5.29. Structured file – Selecting a file

Third, when the source file is a text file, it should be specified how rows and columns are seperated (Figure 5.30).

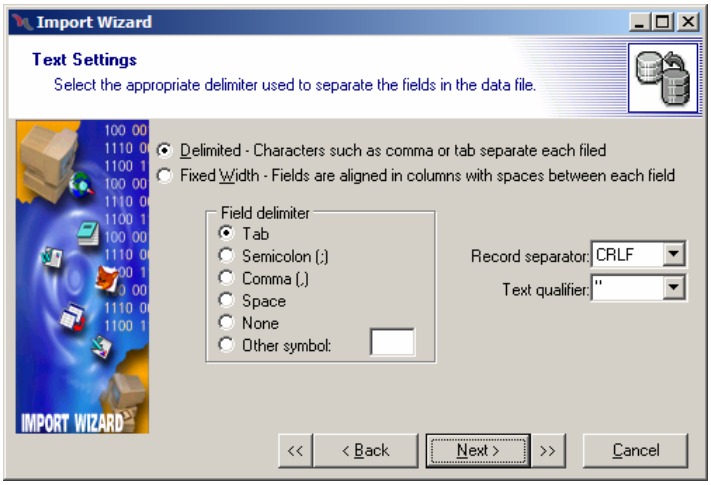


Figure 5.30. Structured file – Specifying text file properties

When the file is specified as fixed width, the columns should be specified manually (Figure 5.31).

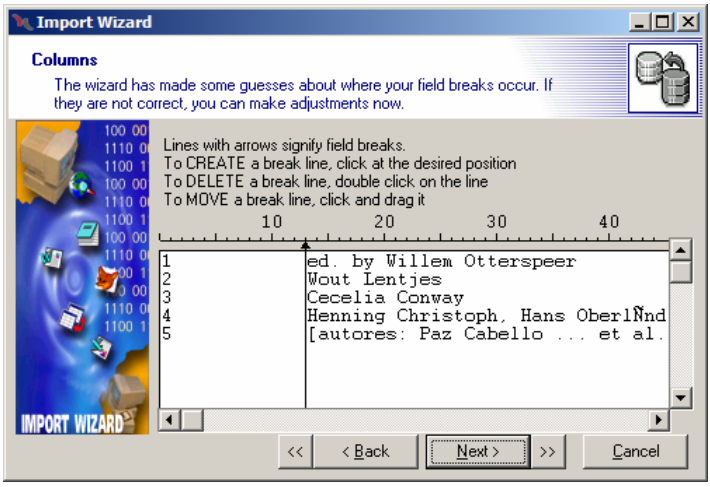

Import Wizard
Columns
The wizard has made some guesses about where your field breaks occur. If they are not correct, you can make adjustments now.
Lines with arrows signify field breaks.
To CREATE a break line, click at the desired position
To DELETE a break line, double click on the line
To MOVE a break line, click and drag it
10
20
30
40
1 ed. by Willem Otterspeer
2 Wout Lentjes
3 Cecelia Conway
4 Henning Christoph, Hans Oberl
5 [autores: Paz Cabello ... et al.
IMPORT WIZARD
<< < Back Next > >> Cancel

Figure 5.31. Structured file – Specifying fixed-width text file columns

Fourth, additional properties must be specified (see for an example Figure 5.32). The first and last row to get the data from must be specified, as must the format of date fields, time fields, and numbers. If the first row is specified as being larger than 1, it can also be specified what row contains the field names.

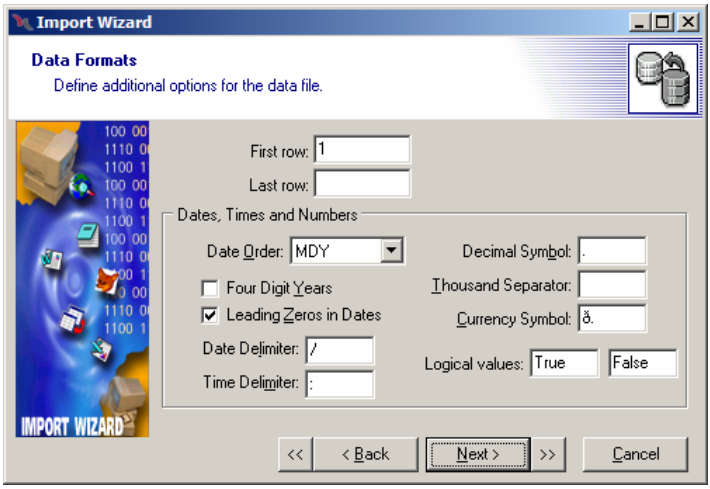


Figure 5.32. Structured file – Specifying file properties

Fifth, a preview of the file is shown (Figure 5.33). If the data does not show correctly, the settings can be changed by pushing the `Back` button. In the file preview window, the upper row shows the fieldnames that will be shown as source fields in the tag settings (see section 5.6.11 on page 64). You can click on the upper row to change the field names of columns.

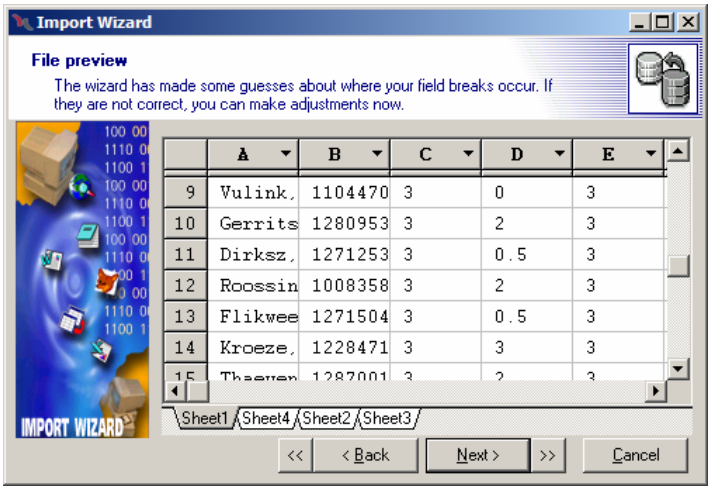

Import Wizard
File preview
The wizard has made some guesses about where your field breaks occur. If they are not correct, you can make adjustments now.
IMPORT WIZARD
A
B
C
D
E
9 Vulink, 1104470 3 0 3
10 Gerrits 1280953 3 2 3
11 Dirksz, 1271253 3 0.5 3
12 Roossin 1008358 3 2 3
13 Flikwee 1271504 3 0.5 3
14 Kroeze, 1228471 3 3 3
Sheet1 Sheet4 Sheet2 Sheet3
<< < Back Next > >> Cancel

Figure 5.33. Structured file – Previewing the file

Lastly, the the wizard will close. The fields in the file can now be mapped to tags, which will be described in section 5.6.11.

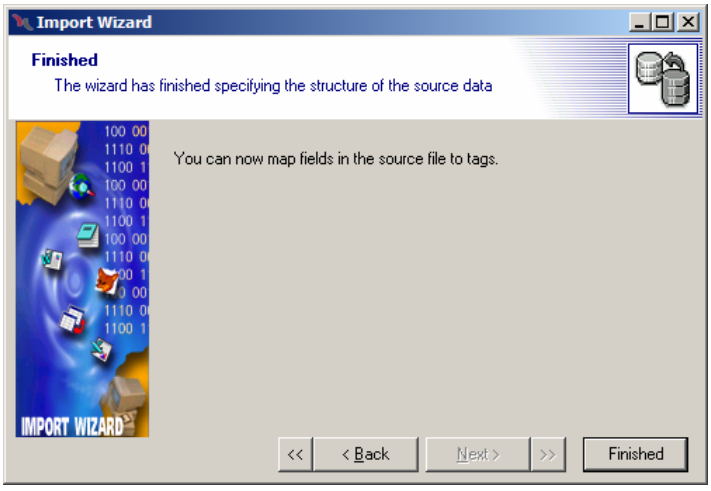

Import Wizard
Finished
The wizard has finished specifying the structure of the source data
IMPORT WIZARD
You can now map fields in the source file to tags.
<< < Back Next > >> Finished

Figure 5.34. Structured file – Closing the wizard

5.6.9 Indexing Outlook Email messages

The *CollectionConnection* Indexer can index Email messages from Microsoft Outook. The Outlook folders that should be indexed can be specified in the Queries and Tags window (see Figure 5.35). Note that selecting a folder will not automatically select it's subfolders! So, subfolders should each be selected individually. Furthermore, the profile to access Outlook should be specified when necessary. Note that, during indexing, *CollectionConnection* can ask for a password multiple times when an Outlook file is password protected.

Attachments can be searched too. All files in the attachments will be put through the Ifilter interface. This means that in Microsoft Word documents, Microsoft Powerpoint documents, and Microsoft Excel documents can be indexed by default. When the PDF Ifilter from adobe is installed, PDF files can be indexed too. Furthermore, when the *CollectionConnection* Ifilter for zip-files is installed (see Chapter 4 on page 27), files in attachments that are compressed with Zip will be decompressed after which each individual file is put through the Ifilter interface.

Oulook must be installed as the default E-mail client on the computer!

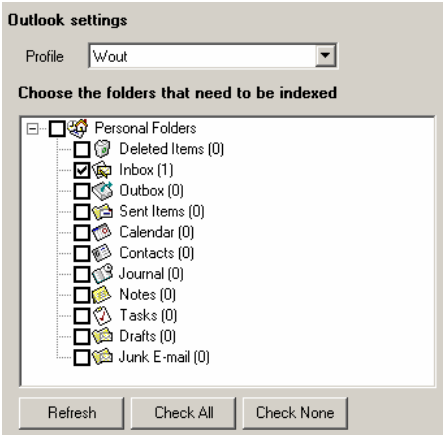


Figure 5.35. Indexing Outlook Email messages

The following fields are available for each E-mail message:

- Sender name
- Sender E-mail address
- Subject
- Date/time
- Recipient name
- Recipient E-mail address
- E-mail account
- Priority
- Outlook Express folder
- Text contents.
- Attachments contents
- Source E-mail. This will embed the E-mail message itself as a binary attachment in the Index. The datatype should be specified as binary! This will allow the message to be extracted from the index and opened by Outlook without the original email.

These fields can be mapped to tags in the tag settings window, see section 5.6.11.

5.6.10 Indexing Outlook Express Email messages

The *CollectionConnection* Indexer can index Email messages from Microsoft Outook Express version 5 or version 6. The Outlook Express folders that should be indexed can be specified in the Queries and Tags window (see Figure 5.36). Additionally, the path that contains the Outlook Express files can be specified. If the path is not specified explicitly, the default path for Outlook Express is used.

Attachments can be searched too. All files in the attachments will be put through the Ifilter interface. This means that in Microsoft Word documents, Microsoft Powerpoint documents, and Microsoft Excel documents can be indexed by default. When the PDF Ifilter from adobe is installed, PDF files can be indexed too. Furthermore, when the *CollectionConnection* Ifilter for zip-files is installed (see Chapter 4 on page 27), files in attachments that are compressed with Zip will be decompressed after which each individual file is put through the Ifilter interface.

Oulook Express may not be active when an index is created!

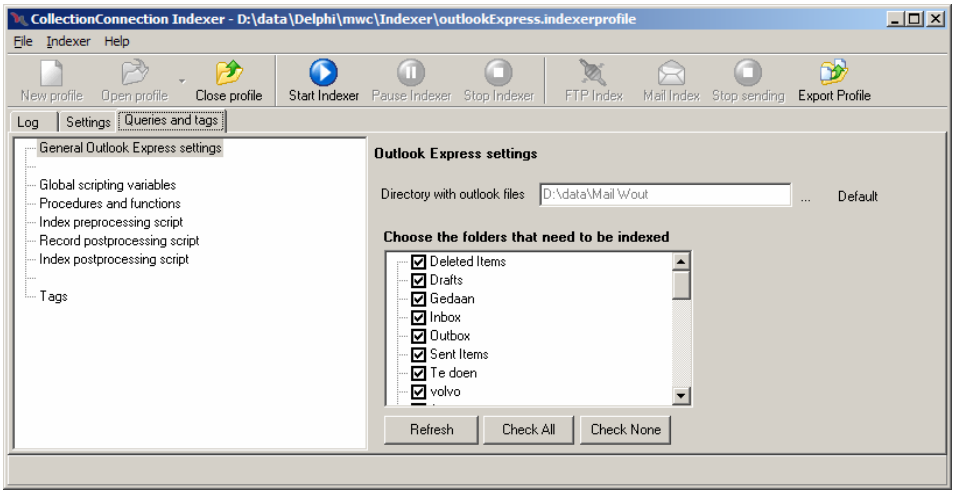


Figure 5.36. Indexing Outlook Express Email messages

The following fields are available for each E-mail message:

- Sender name
- Sender E-mail address
- Subject
- Date/time
- Recipient name
- Recipient E-mail address
- E-mail account
- Priority
- Outlook Express folder
- Text contents
- Attachments contents
- Source E-mail. This will embed the E-mail message itself as a binary attachment in the Index. The datatype should be specified as binary!

This will allow the message to be extracted from the index and opened by Outlook Express without the original email.

These fields can be mapped to tags in the tag settings window, see section 5.6.11.

5.6.11 Tags

The tags provide the actual mapping by specifying the source of the tag (e.g., field in a query), and some other settings. Figure 5.38 shows the overall screen with settings. The specific parts will be discussed respectively.

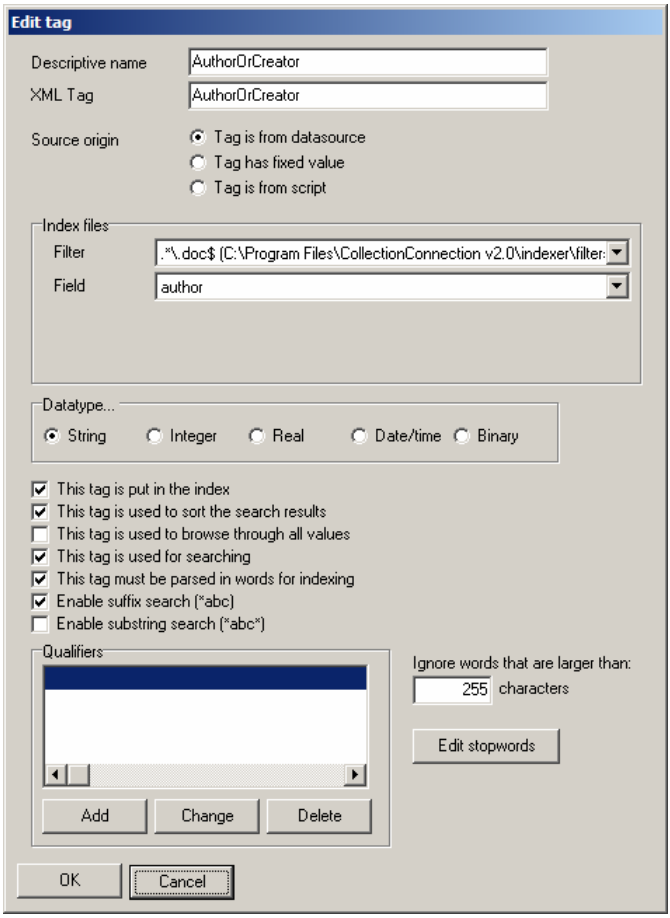


Figure 5.37. Specifying tags

5.6.11.1 Generic settings

Each tag has a descriptive name (Figure 5.38). This name is solely used in the indexer profile to identify it. Furthermore, each tag must specify how it is named in the XML-records. The distinction between the descriptive name and the XML tag is made because an XML-record can contain multiple tags with the same name. For example, the XML-tag 'title' can be based on both the field 'title' from the query 'titles' and on the field 'public title' from the query 'objects'.

Figure 5.38. Tag name

There are three ways in which a tag can get it's information (Figure 5.39). First, it can get it's information directly from the datasource (a database field, an xml tag, a file filter, etc.). Second, a script can be used to determine the tag, and third, a tag can have a fixed value.

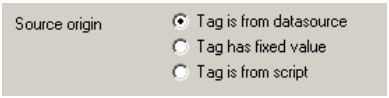


Figure 5.39. Tag source

If the tag source is a database field (Ado/Oracle), then the following items need to be specified (Figure 5.40):

1. The query
2. The objectID field in the query (only in multi-table indexer profiles)
3. The field that is the actual source for the tag

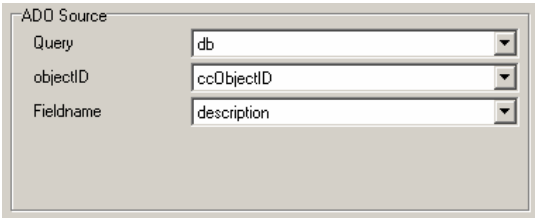


Figure 5.40. Tag source (Ado/Oracle)

If the tag source is an XML file, the source tag including attributes must be specified (Figure 5.41).

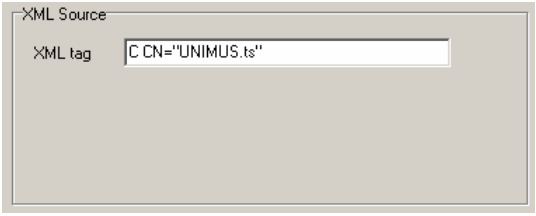


Figure 5.41. Tag source (XML)

If the tag source is an Adlib file, the source record specifier must be specified (Figure 5.42).

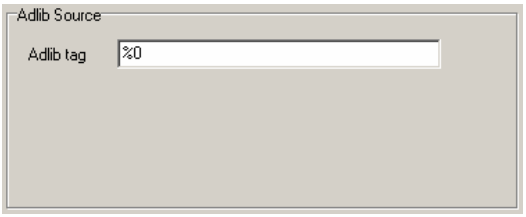


Figure 5.42. Tag source (Adlib)

If the tag source is a file, the filter and the filter's field must be specified (Figure 5.43).

Figure 5.43. Tag source (File)

If the tag source is a script, then the script must fill either the string variable `ccTag` or the array of strings `ccTagList`:

Example with `ccTag: ccTag="test"`
Example with `ccTagList`:

```
ccTagList.Add("test")
ccTagList.Add("another test")
```

In the latter example, the XML-record will contain two tags. Scripts must be programmed in Basic, see Appendix A. Scripts can get to the underlying data source by using the following variables (see Figure 5.44 for an example):

1. Ado/Oracle

   - `dataset`. This variable is a dataset. The query in the tag-window specifies what query the script will use to build the tag data. The function `fields` can be used to get to the individual fields of the query; the return type is a `Variant` and type conversion will take place automatically. Example:

     ```
     ccTag="NAAM:"+dataSet.fields("naam")
     ```

2. *XML*

   - `ccXMLSourceRecord.` This variable is a string that contains the whole source XML record. The record can be processed in the script with the Microsoft XML parser. For example, the following script concatenates the identifier and title of an XML record:

```
dim xmld
xmld = Create0le0bject("microsoft.xmldom")
xmld.loadxml(ccXMLSourceRecord)
ccTag=xmld.documentElement.selectsinglenode("identifier")
  .text+"-"+
  xmld.documentElement.selectsinglenode("title").text
xmld = nothing
```

3. Adlib:

   - `ccAdlibSourceRecord`. This variable is a string that contains the whole source Adlib record.
   - `getFieldFromAdlibRecordCount(adlibField)`. This function returns the number of rows in the source Adlib-record that are of the specified `adlibField`-type.
   - `getFieldFromAdlibRecord(adlibField, occurrence)`. This function will return a field from the Adlib record. `Adlibfield` is the field specifyer, `occurrence` denotes the n^{th} occurrence of the field in the record.

4. Files:

   - `ccFileName`: This variable contains the filename of the file that will be indexed. With the function "`extractFilePath`", the filename can be truncated to its path.
   - `ccFileArchive`: Boolean variable that specifies whether the archive bit is set.
   - `ccFileHidden`: Boolean variable that specifies whether the hidden bit is set.
   - `ccFileRead0nly`: Boolean variable that specifies whether the read only bit is set.
   - `ccFileSystem`: Boolean variable that specifies whether the system file bit is set.
   - `ccFileSize`: Integer variable that specifies the size of the file.
   - `ccFileModifiedDate`: Date/time when the file was last modified. This is a floating point variable. The integral part is the number of days that have passed since 12/30/1899. The fractional part is the fraction of a 24 hour day that has elapsed. Use the `dateTimeToStr` to convert it to a string.
   - `ccFileCreateDate`: Date/time when the file was created. This is a floating point variable. The integral part is the number of days that have passed since 12/30/1899. The fractional part is the fraction of a 24 hour day that has elapsed. Use the `dateTimeToStr` to convert it to a string.
   - Example script (the script will write file characteristics to the log window during indexing):

```
showline(ccFilename)
if ccFileArchive then showline("  archive") end if
if ccFileRead0nly then showline("  readonly") end if
if ccFileSystem then showline("  system") end if
if ccFileHidden then showline("  hidden") end if
```

```
showline("  size: "+intToStr(ccFileSize))
showline("  created: "+dateTimeToStr(ccFileCreateDate))
showline("  modified:
    "+dateTimeToStr(ccFileModifiedDate))
```

5. Outlook:

   - `ccSenderName`
   - `ccSenderEmail`
   - `ccSubject`
   - `ccReceivedDate`
   - `ccReceiverName`
   - `ccReceiverEmail`
   - `ccAccount`
   - `ccPriority`
   - `ccImportance`
   - `ccSensitivity`
   - `ccMessageSize`: this is an `Integer` variable specifying the size including attachments.
   - `ccFolderName`: the folder the message is in.
   - `ccBodyContents`: the text contents of the body of the E-mail message.
   - `ccAttachmentsContents`: the text contents of all attachments.

6. Outlook Express:

   - `ccSenderName`
   - `ccSenderEmail`
   - `ccSubject`
   - `ccReceivedDate`
   - `ccReceiverName`
   - `ccReceiverEmail`
   - `ccAccount`
   - `ccPriority`
   - `ccFolderName`: the folder the message is in.
   - `ccBodyContents`: the text contents of the body of the E-mail message.
   - `ccAttachmentsContents`: the text contents of all attachments.
   - `ccSourceEmail`: the `MIME-RFC-822` formatted E-mail message.

Additionally, scripts for all datasources can access the variable `ccRecordNumber`, that returns a sequential number for each XML-record. It can be used as a unique identifier.

7. Structured files:

   `dataset`. This variable is a dataset. The function `fields` can be used to get to the individual fields of the current row of the structured file; the return type is a `Variant` and type conversion will take place automatically. The field names are the one specified in the structured file preview window (see section 5.6.8 on page 60). For example:
   ```
   ccTag="NAAM:"+dataSet.fields("naam")
   ```

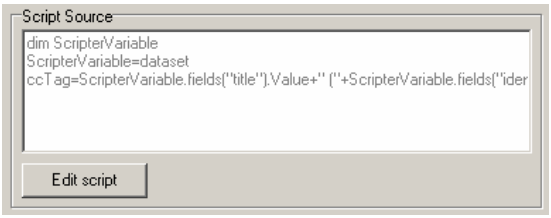


Figure 5.44. Sample Script source

By pressing the "edit script" button, the script can be changed.

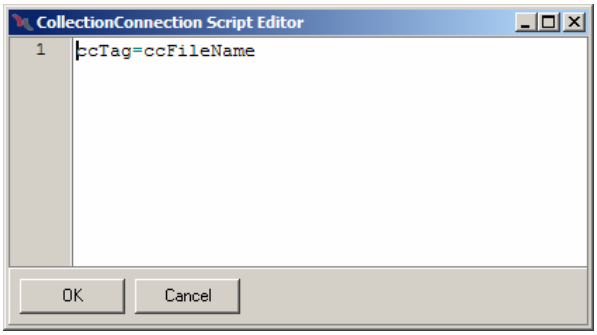


Figure 5.45. Script editor

If the tag source is a fixed value, this value can simply be typed in (Figure 5.46). All XML-records in de *CollectionConnection* index get this value for the tag. This can be used for distributed searching or combining indexes.

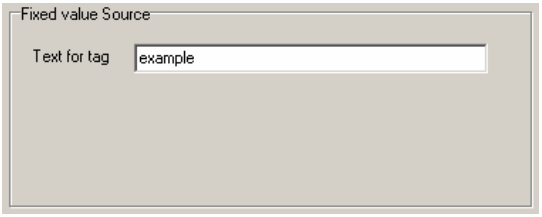


Figure 5.46. Fixed value

5.6.11.2 Datatype

CollectionConnection knows 5 datatypes (Figure 5.47). The datatype determines the way in which result-sets are sorted and the operators that can be used during searching:

- **String**. String (which is normal text) is sorted alphabetically. It is possible to perform postfix, suffix, and substring searches (see Section 5.6.11.3 on page 71).
- **Integer**. Integers are whole numbers and are sorted on their numerical value. It is possible to search for equallity (=), larger than (>), smaller than (<), larger or equal (=>), and smaller or equal (<=).

- **Real**. Reals are numbers with a fractional part, e.g., 3.1415. Sorting and searching are the same as with Integers.
- **Date/time**. Date/time fields should specify a valid date and time in the following format: dd-mm-yyyy hh:mm:ss. Sorting and search operators are the same as with Integers and Reals.
- **Binary**. Binary contents is MIME64-encoded in the tags. A client must decode it to get back the original binary data. It is good practice to add a qualifier to the tag stating the filename and/or Mime-type of the binary data (see section 5.6.11.4 on page 72).
- **Script**. With this datatype, the contents of the tag is determined on the *CollectionConnection* server by executing a script (for scripts see also Appendix A). When a record is requested from the server, the script in the tag is executed and the result of the script is put in the contents of the tag. The source of the script is typically a fixed value, but it can also be stored in the source data, or created by a script itself. An example to use server side scripts is to get stock positions in an E-commerce application. Typically, the product portfolio is rather stable but stock positions fluctuate. The stock positions can be retrieved from a database by the server when a record is requested. In this way, the advantages of both worlds (efficiency for full text search on product descriptions, and up-to-date stock positions) are met. The source XML record is in the variable `ccXMLRecord`, and the result of the script must be put in the variable `ccTag`. The following example shows how to get data from a database on the basis of one of the tags in the record:

```
dim xmld,identifier,0bjConnC,0bjRSC
xmld=Create0le0bject("microsoft.xmldom")
xmld.loadxml(ccXMLRecord)
identifier=xmld.documentElement.selectsinglenode("identif
    ier").text
0bjConnC=Create0le0bject("AD0DB.Connection")
0bjConnC.0pen("Driver={Microsoft Access Driver
    (*.mdb)};DBQ=c:\data\authors.mdb")
0bjRSC=Create0le0bject("AD0DB.RecordSet")
0bjRSC.ActiveConnection=0bjConnC
0bjRSC.0pen("Select * from 0bjects where
    identifier='"+Identifier+"'")
if not 0bjRSC.E0F then
  ccTag=0bjRSC.Fields("stock").Value
end if
0bjRSC.Close
0bjConnC.Close
```

Since the value of script tags is determined dynamically, the result of the scripts can not be searched, and the tag can not be sorted on.

Poorly designed scripts can deteriorate the performance of the server. They can even cause the server to hang!

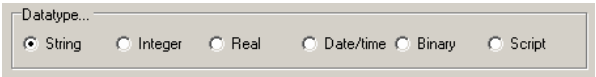


Figure 5.47. Datatype

The datatype in the *CollectionConnection* does not have to be the same as in the source data; the data will be converted automatically when possible. For example, an integer field in the database can be a string field in the Indexer. When converting strings to date/time, date format in the source data should be `dd-mm-yyyy`, and the time format should be `hh:mm`.

5.6.11.3 Indexing options

Sorting and full text searching increase the index size. For that reason, the allowed sort and search operators can be specified for each tag individually (Figure 5.48Figure).

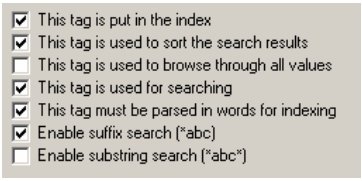

This tag is put in the index
This tag is used to sort the search results
This tag is used to browse through all values
This tag is used for searching
This tag must be parsed in words for indexing
Enable suffix search (*abc)
Enable substring search (*abc*)

Figure 5.48. Indexing options

Explanation of options:

- The tag will be put in the index. If this is disabled, the tag can be searched but it will not be part of the index itself. This is especially useful for searching large fields (such as the contents of files) that do not have to be shown. NOTE: tags that are not in the index can not be searched anymore in a combined index!

- It can be chosen whether the tag can be used to sort the result set.

- Use the tag to browse through all values. If this option is enabled, all distinct values of a tag are stored separately. These values can then be browsed. This is similar to a SQL distinct query. For example, the query "select distinct author from books" results in a list with all authors.

- The tag can be used for searching. If this is disabled, the tag will be in the record but no searching can take place on the tag. If searching is enabled, then the following options are available:

    a. Parsing the tag. For full text search, the contents of the tag is parsed in its individual words. Sometimes, this is not desirable, for example in an identifier field in which the individual parts have no meaning (e.g., the identifier 'AFE 323-32').
    b. Suffix search. By default, all tags allow postfix search. For example, a search for '`tel*`' would find records that have a word that starts with '`tel`'. Suffix search is configurable. With suffix search, searching for '`*tel`' will find records with words that ends with '`tel`'

c. Substring search. Searching for '`*tel*`' finds records with words that have '`tel`' somewhere, e.g. '`motels`'. Substring search on large text fields will significantly increase the index size!

- Maximum word length. To preserve performance, the maximum word length can be specified. This does not have to be used to limit the length of normal words, but can be used to prevent the indexer trying to index binary contents without any delimiters (e.g., when a BLOB database field is accidentally mapped to a string field). The default maximum word length is 255.

5.6.11.4 Attributes

XML tags can contain attributes. Attributes are used to specify the state of the tag, for example:

- `<description language="english">This is the English description</description>`
- `<description language="dutch">Dit is de nederlandstalige beschrijving</description>`

Attributes can be specified as plain text, but the value of attributes can also be retrieved from the underlying data source (Figure 5.49).

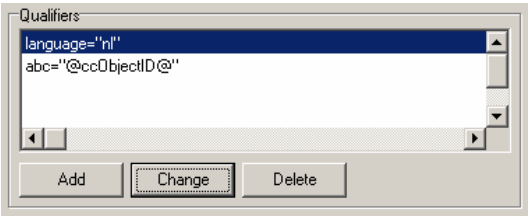


Figure 5.49. Tag attibutes

Examples:

- `language="NL"`. All tags will have the same attribute.
- `Language="@lng@"`. The value part of the attribute will be retrieved from the datasource:

  a. With ADO-based datasources, there must exist a field '`lng`' in the query.
  b. With Adlib-based datasources, if the record contains a field specified as '`lng`', the value will be retrieved from that field.
  c. With XML-based datasources, there must be a tag '`lng`' in the record.

The following attribute-specifiers can be used during file-indexing:

a. `@ccfilename@`: the name of the file
b. `@ccfilepath@`: the path of the file
c. `@ccfileext@`: the file's extension

d. `@ccfilemimetype@`: the mimetype. The indexer will try to determine the mimetype on the basis of the extension of the file. This can be useful when files of different kinds are indexed, and the files are downloaded by ASP-pages. The ASP page can now also report the mimetype of the file to the browser, that can then use the appropriate viewer.

Example:

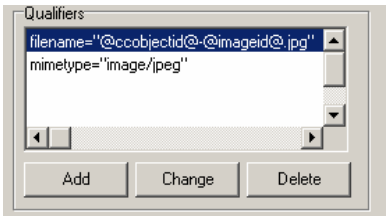


Figure 5.50. Qualifiers for file indexing

This example adds the attributes filename and mimetype to a binary tag that contains the contents of a Blob field from a database. The filename is specified by two fields from the query that the Blob field is in.

Note that the value part of an attribute must be enclosed with double quotes, otherwise the resultset will contain invalid XML.

5.6.12 Stopwords

Indexing large bodies of text (for example, the full contents of a large number of books) can result in a large index and poor search performance. This can be reduced by specifying commonly used words that do not add much to the meaning or content of the text. Such words will be ignored both during the indexing process in the indexer and during searches by the server. Stopwords can be specified for each tag individually.

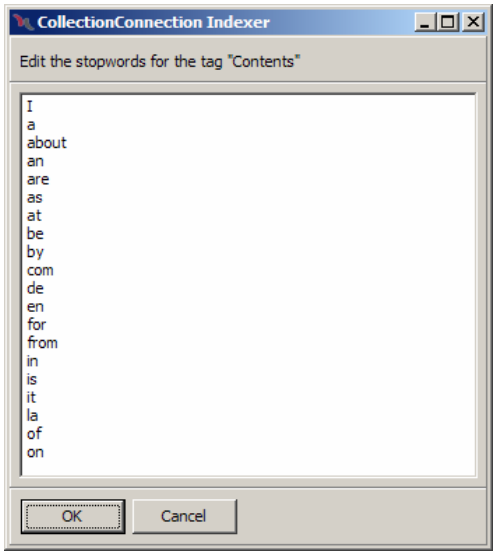


Figure 5.51. Tag stopwords

5.6.13 Dynamic Content Links

Dynamic Content Links provide the means to relate XML records to each other or to external entities without including such links explicitly in the source text. The following examples will clarify this:

- Descriptions often contain concepts that are explained in a thesaurus. For example, the phrase "Ming dynasty" in the description "This object is probably created in the Ming dynasty", is explained in the thesaurus. Only, the author of the description did not create this link manually. *CollectionConnection* can do this automatically.
- Object descriptions can refer to each other by specifying each others identifiers. Example: "This object was found simultaneously with object 123-ABC. Again, the author will not have created a reference to a URI himself since such a reference is context dependent.
- Other examples are object descriptions that contain titles, names of collections, names of collectors or authors, etc.

In the Indexer, a Dynamic Content Link is specified with the following elements (Figure 5.52):

- Descriptive name. This is the internal name for the Indexer.
- Query. The query that provides the information for the Dynamic Content Link when the datasource is ADO or Oracle based.
- Text to link. This is a field in the query that specifies the text in the XML-record that is to be referenced. For example, "Ming dynasty".
- Link type name. A record can contain multiple kinds of Dynamic Content Links simultaniously, e.g., for the thesaurus, for titles, for authors, etc. In order to be able to separate these, Dynamic Content Links have a name.
- Link value. This is the reference itself. For example, the "Ming dynasty" would refer to "aat.bac.aba.bbd".

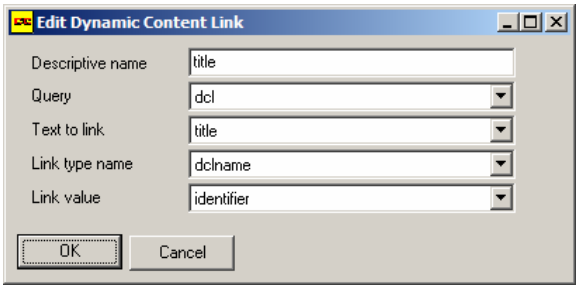


Figure 5.52. Dynamic Content Link settings

A dynamic content link is added to an XML-record in the resultset when requested. The following XML-tag shows an example:

```
<ccdcl name="thesaurus" text="Ming
dynasty">AAT.BAC.ABA.BBD</ccdcl>
```

A record can contain multiple of such tags, each specifying a Dynamic Content Link. Typical uses are:

- Make href's in the HTML text: search for "Ming Dynasty" in the XML record, put '`<A href='thesaurus.asp?node=AAT.BAC.ABA.BBD'>`' before it, and '`</A>`' after it. If the uses now clicks on the link, it is automatically forwarded to the node in the thesaurus. The *CollectionConnection* Client (see Chapter 8) contains functionality to do this.
- Provide a list with relevant concepts in the record's tag. Much the same as above, but now links are not created in the text itself but rather after the text in a special 'relevant-links' section.

Server

6 CollectionConnection Server

6.1 Introduction

The *CollectionConnection* Server returns a number of XML-records on the basis of a request. The request can be about collections, objects in a collection, the thesaurus structure, objects classified somewhere in the thesaurus, and a search query on the basis of keywords. The Server can be approached with:

- *A dedicated TCP/IP protocol*. This is useful for writing a client. (An example of such a client is discussed in chapter 8). The advantage of this protocol is that the connection can be kept alive between search requests. This increases the performance.
- *HTTP-protocol for XML output*. This can be used for an XML-browser (e.g., Internet Explorer).
- *HTTP-protocol for HTML output*. With this connectivity, the server functions as a fully featured web server, including support for ASP-pages.
- *HTTP-protocol for images*. Web-browsers can request an image from the Image server (see also chapter 11).
- *SOAP-protocol*. This enables SOAP enabled development environments (such as Microsoft .Net) to query *CollectionConnection* easily.
- *HTTP-protocol for the Open Archives Initiative*. The Open Archives Initiative (OAI) is a protocol for harvesting purposes. *CollectionConnection* fully implements the protocol. See `http://www.openarchives.org`.
- *HTTP-protocol for search engine spiders*. This protocol provides an efficient way to let search engines such as Google and Alta Vista include the information from the index.
- *Z39.50 protocol*. Z39.50 is used all over the world to disclose bibliographic information. Z39.50 clients such as Bookwhere can connect to *CollectionConnection* and search through the database. For more information about Z39.50, see http://www.loc.gov/z3950/agency/.
- *FTP-Protocol*. Indexes, images, and website contents can be sent to the server with an FTP-client.

All communication (except FTP) can be secured with a secure socket layer or SSL. The server certificate that is needed is included in the license file. Furthermore, access to each port can be secured by requiring a username and password. The actual authentication depends on the protocol and will be discussed shortly.

There are three versions of the *CollectionConnection* server:

- *The normal server*. This server is typically used to provide on-line access to the collection index.
- *The browser*. The browser is typically used to browse through a collection locally. To be more specific, the browser contains both an internet browser and the normal server, where the former connects to

the latter (i.e., the browser part of the Server connects to the server part of the Server).

- *The fullscreen browser*. This is the same as the browser, but it is shown full screen without a navigation toolbar. This is useful for information kiosks.

Note that the browser and the full screen browser are exactly the same as the normal server, except that for the former two, the settings can not be specified (i.e., are not shown). Therefore, the browser (and full screen browser) are perceived by the user as being only a browser, and the user is not bothered with all kinds of settings. The browser and full screen browser may be freely distributed with an index. The normal server may not be freely distributed.

From here on, we will only discuss the normal server (that, incidentally, also contains a browser window). The browser pane uses the Microsoft Internet Explorer control to connect to the server via the HTTP-HTML port. The default external web browser can be started by clicking on the 'start external browser' button. If the browser pane is empty when started, the 'home' button should be clicked (the button with the little house). The browser and full screen browser shield the user from the server settings by showing only the browser window. If the browser is started, it searches for the first server profile in the program directory of the Server, loads the profile, and starts the servers. The user does not notice this. The browser can be used to distribute a part of the collection on CD or on kiosks without the necessity to have IIS/PWS installed, while at the same time allowing the presentation of the collection to be designed and implemented using HTML and ASP.

6.2 The system

6.2.1 Creating a new profile

Creating a new profile consists of a number of steps (Figure 6.1). First, a license has to be loaded. This license is provided by the reseller. Second, a number of settings can be specified. These settings can be left empty to be filled in later. Third, the profile must be given a name. The name of the profile will also be the name of the index that is used.

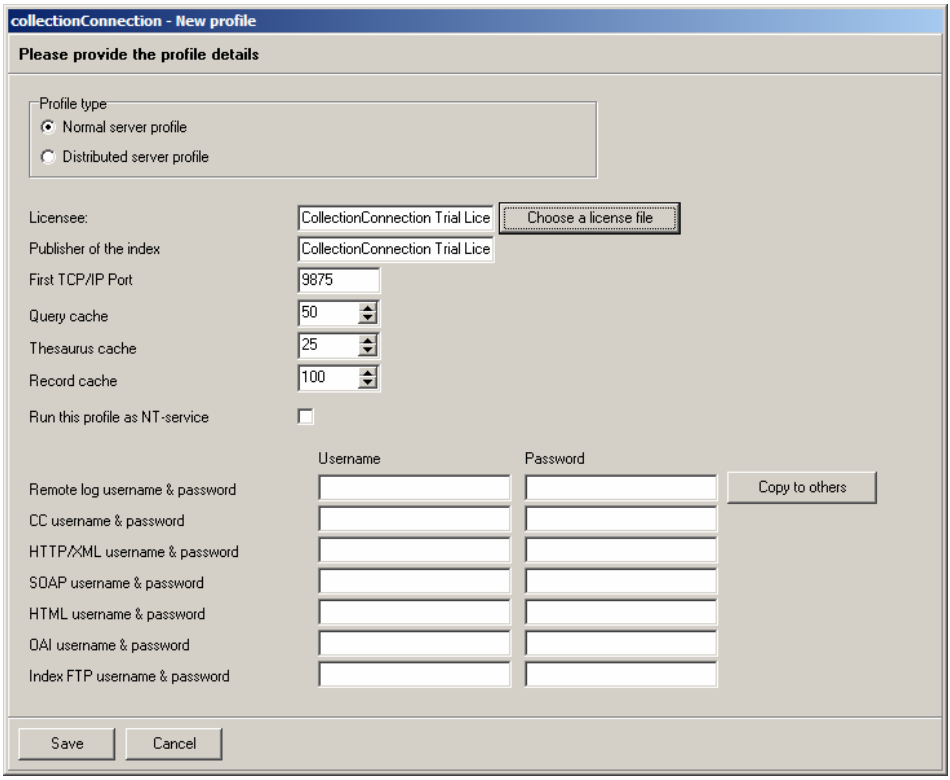


Figure 6.1. Creating a new server profile

The settings will be discussed in the following sections, but two settings are worth mentioning here: First, the first TCP/IP port. Each protocol uses it's own port, so, depending on the configuration, several ports might be needed. By default, *CollectionConnection* will generate a consecutively numbered list of ports (starting with the one specified in the 'new profile' dialog) and assigns them to the respective protocols. In the settings of the individual protocols, these ports can be individually changed afterward. Second, there is a button to copy the username and password that are filled in with the Remote Log protocol to the other usernames and passwords. Generally, it is not wise to use the same username and password for all protocols. For example, a HTML username that is used to view the information might not be given to the same persons that must update this information with the FTP username and password.

6.2.2 Loading and saving a profile

A profile can be given as command line parameter when the program is started, in which case it will open the specified profile. This can be done by a making a shortcut in Windows, for example with the following command line:

```
D:\program files\collectionconnection\ccServer.exe
"demo.serverprofile"
```

If the command line contains the parameter "`-startservers`", then the servers are started automatically with the profile, otherwise, the profile is just opened. If no

command line parameter is specified, then the user must manually choose a profile (Figure 6.2).

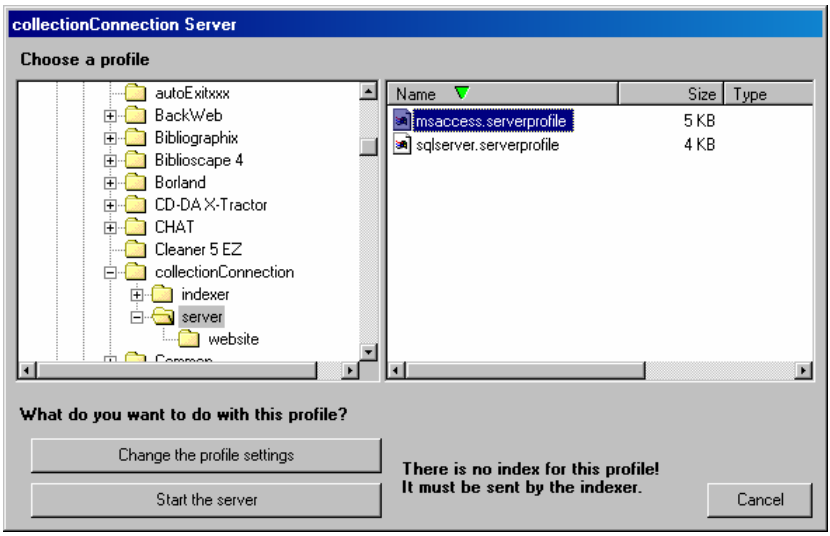


Figure 6.2. Opening a Server profile

The following actions are possible:

- *Change the profile settings*. The settings can be changed and saved (by clicking "save profile" in the main Server screen).
- *Start the server*. The server is started.

If there is not an index available, the server is started anyway and waits until the indexer sends an index. If the Server has received an index, it closes the old index and loads the new index.

The index file is stored in the same directory as the profile.

6.2.3 The Log window

The log window shows the activities that take place (Figure 6.3).

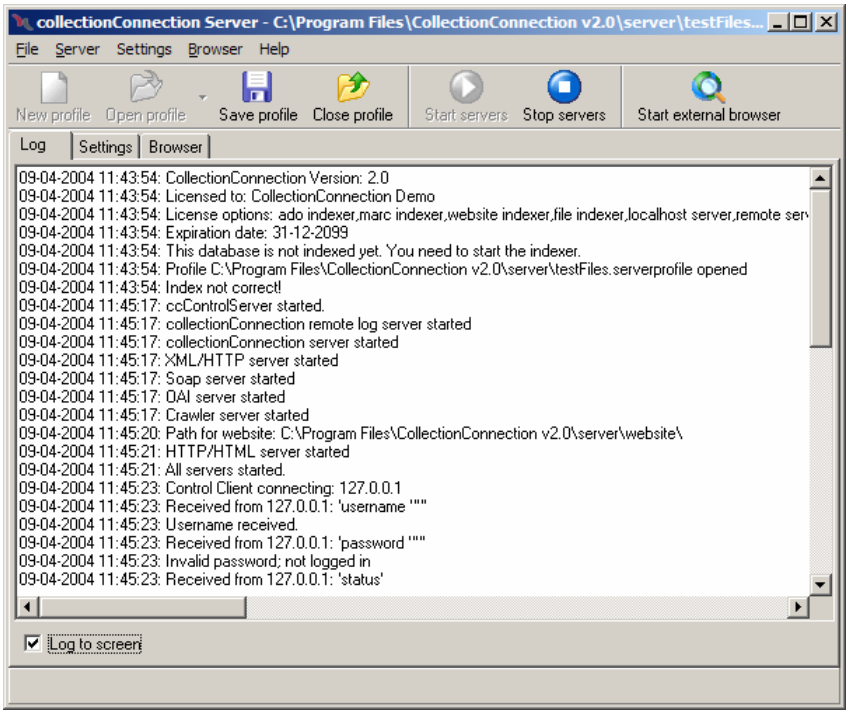

collectionConnection Server - C:\Program Files\CollectionConnection v2.0\server\testFiles...
File Server Settings Browser Help
New profile Open profile Save profile Close profile Start servers Stop servers Start external browser
Log Settings Browser
09-04-2004 11:43:54: CollectionConnection Version: 2.0
09-04-2004 11:43:54: Licensed to: CollectionConnection Demo
09-04-2004 11:43:54: License options: ado indexer,marc indexer,website indexer,file indexer,localhost server,remote ser
09-04-2004 11:43:54: Expiration date: 31-12-2099
09-04-2004 11:43:54: This database is not indexed yet. You need to start the indexer.
09-04-2004 11:43:54: Profile C:\Program Files\CollectionConnection v2.0\server\testFiles.serverprofile opened
09-04-2004 11:43:54: Index not correct!
09-04-2004 11:45:17: ccControlServer started.
09-04-2004 11:45:17: collectionConnection remote log server started
09-04-2004 11:45:17: collectionConnection server started
09-04-2004 11:45:17: XML/HTTP server started
09-04-2004 11:45:17: Soap server started
09-04-2004 11:45:17: OAI server started
09-04-2004 11:45:17: Crawler server started
09-04-2004 11:45:20: Path for website: C:\Program Files\CollectionConnection v2.0\server\website\
09-04-2004 11:45:21: HTTP/HTML server started
09-04-2004 11:45:21: All servers started.
09-04-2004 11:45:23: Control Client connecting: 127.0.0.1
09-04-2004 11:45:23: Received from 127.0.0.1: 'username '"'
09-04-2004 11:45:23: Username received.
09-04-2004 11:45:23: Received from 127.0.0.1: 'password '"'
09-04-2004 11:45:23: Invalid password; not logged in
09-04-2004 11:45:23: Received from 127.0.0.1: 'status'
Log to screen

Figure 6.3. The Server Log window

6.2.4 The Settings window

The server opens *CollectionConnection* to the outside world. Because there is a large number of supported protocols, there also is a large number of settings. The settings window is shown in Figure 6.4. The specific settings are discussed in detail in the following sections.

⚠ After changing the profile, it must be stored by pressing the 'Save profile' button!

6.2.5 General settings

The *CollectionConnection* server has the following general settings:

- *The publisher of the collection.* Each dc-record in the result set will contain a `<publisher>` tag with this information.
- *NT-Service.* The server can be configured to run as NT-service. This option is not available on Windows 98 and ME, only on Windows NT/2000/XP/2003. If this option is checked, the server will run without anyone logged in on the computer, and the server will start during system startup. After checking this, the program can be run as a service immediately by starting the service in the Services applet of the Administration Tools in the Windows control panel.

- *Bind servers to localhost*. The *CollectionConnection* server is a true server. This means that if the server is active when your computer is connected to the Internet, and you don't have a firewall, others can search your index like you can yourself. This is useful when you want to publish your information on a webserver, but probably not when you are working on your home computer with a ADSL internet connection. (Note that the *CollectionConnection* server only provides access to the index and website template. Others will not be able to download files or change anything on your computer if the ftp-servers are disabled.) Access by remote computers can be disabled by checking the `Bind servers to localhost` checkbox. An additional advantage is that there will be no firewall messages when the server is activated on a computer where a firewall is installed.
- *Logging*. System messages can be logged to file. This file can be used to analyze the functioning of *CollectionConnection*.
- *Cache settings*. Several items can be cached, which costs memory but improves performance. The trade-off mainly depends on the size of the index and can be determined by trial-and-error.

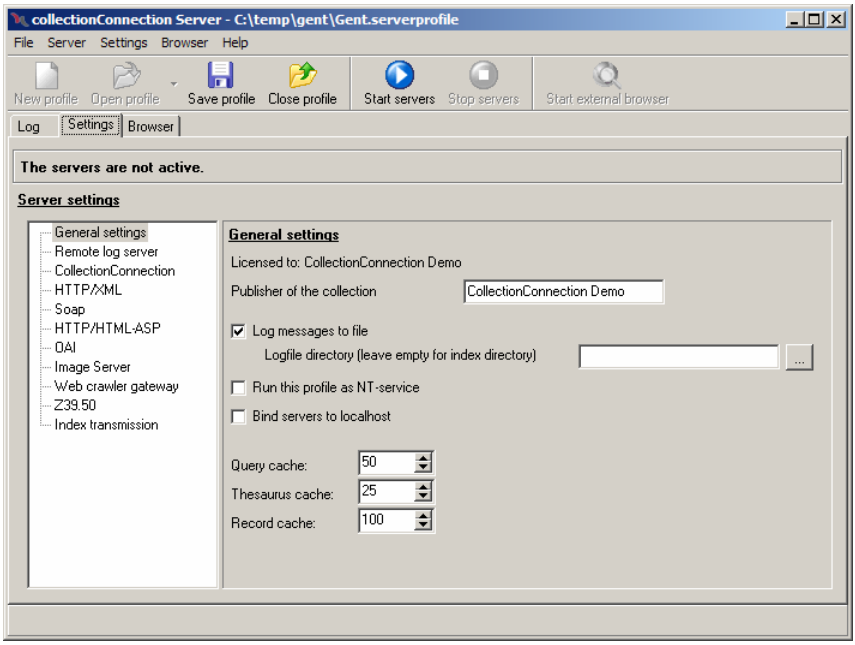


Figure 6.4. General settings for the server

Next to these general settings, there are settings per protocol. The settings for each protocol will be explained in the following sections.

6.2.6 Remote Log Server

The Remote Log Server is used to view the log remotely. Figure 6.5 shows the settings that need to be configured. It can be enabled or disabled. When enabled, it must be

determined whether the communication is secured with SSL or not. Furthermore, the TCP/IP port must be specified. Lastly, it must be determined whether a client needs to authenticate first.

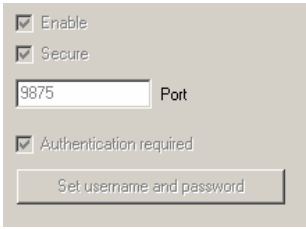


Figure 6.5. Settings for the Remote Log Server server

The Remote Log Viewer (see Figure 6.6) can be used to view what's happening on the server. When connected, the viewer shows events real time. The server address, port, username, password, and security settings must be specified. When connected, the status of the server can be retrieved; this will give general information about the server's memory, harddisk size, CPU speed, and what *CollectionConnection* protocols are active.

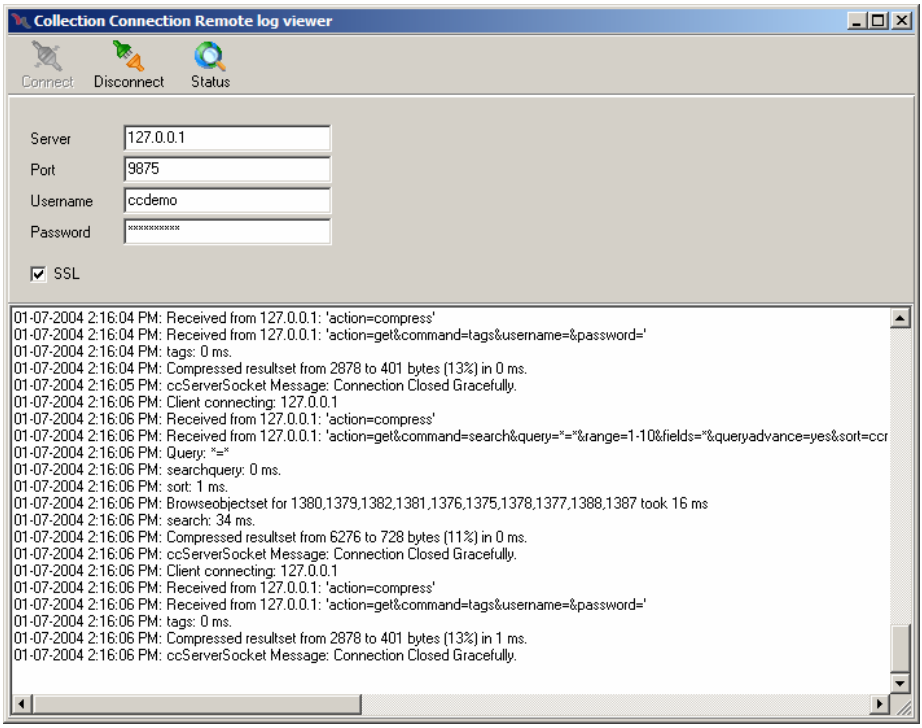

Collection Connection Remote log viewer
Connect
Disconnect
Status
Server
127.0.0.1
Port
9875
Username
ccdemo
Password
SSL
01-07-2004 2:16:04 PM: Received from 127.0.0.1: 'action=compress'
01-07-2004 2:16:04 PM: Received from 127.0.0.1: 'action=get&command=tags&username=&password='
01-07-2004 2:16:04 PM: tags: 0 ms.
01-07-2004 2:16:04 PM: Compressed resultset from 2878 to 401 bytes (13%) in 0 ms.
01-07-2004 2:16:05 PM: ccServerSocket Message: Connection Closed Gracefully.
01-07-2004 2:16:06 PM: Client connecting: 127.0.0.1
01-07-2004 2:16:06 PM: Received from 127.0.0.1: 'action=compress'
01-07-2004 2:16:06 PM: Received from 127.0.0.1: 'action=get&command=search&query=*=*&range=1-10&fields=*&queryadvance=yes&sort=ccr
01-07-2004 2:16:06 PM: Query: *=*
01-07-2004 2:16:06 PM: searchquery: 0 ms.
01-07-2004 2:16:06 PM: sort: 1 ms.
01-07-2004 2:16:06 PM: Browseobjectset for 1380,1379,1382,1381,1376,1375,1378,1377,1388,1387 took 16 ms
01-07-2004 2:16:06 PM: search: 34 ms.
01-07-2004 2:16:06 PM: Compressed resultset from 6276 to 728 bytes (11%) in 0 ms.
01-07-2004 2:16:06 PM: ccServerSocket Message: Connection Closed Gracefully.
01-07-2004 2:16:06 PM: Client connecting: 127.0.0.1
01-07-2004 2:16:06 PM: Received from 127.0.0.1: 'action=compress'
01-07-2004 2:16:06 PM: Received from 127.0.0.1: 'action=get&command=tags&username=&password='
01-07-2004 2:16:06 PM: tags: 0 ms.
01-07-2004 2:16:06 PM: Compressed resultset from 2878 to 401 bytes (13%) in 1 ms.
01-07-2004 2:16:06 PM: ccServerSocket Message: Connection Closed Gracefully.

Figure 6.6. Remote Log Viewer

6.2.7 CollectionConnection

The *CollectionConnection* protocol is a stateful protocol. Once connected, a client can send multiple requests. The following list shows an example of a TCP-IP session (note that the format for input and output will be discussed later):

- Client connects to server
- Server answers with "`Welcome to collectionConnection Server. Publisher: CollectionConnection Demo`"
- Client sends: `action=get&command=search&query=*=*&range=1-10&fields=*`
- Server answers with the requested records.
- Client sends: `exit`
- The server terminates the connection.

6.2.8 HTTP/XML

The HTTP/XML server provides raw HTML with the HTTP protocol. This can be used to show the raw contents of a query in an XML browser (for example, Internet Explorer). Figure 6.7 shows an example using the following URL (note that the query parameters are explained in section 6.3):

- `http://localhost:9877/action=get&command=search&query=*=*&range=1-10&fields=*`

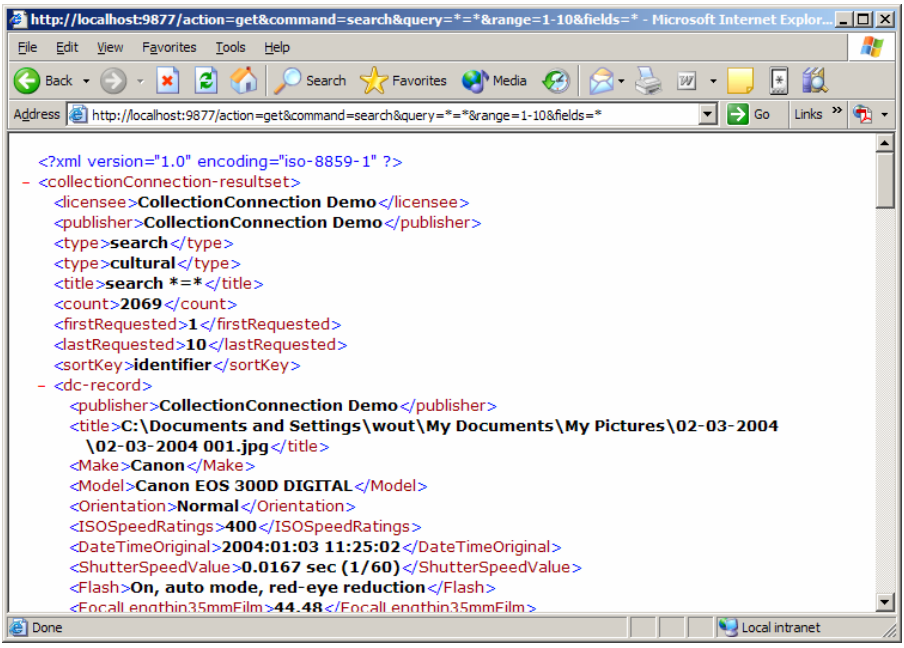


Figure 6.7. View XML in the browser

The settings of the HTTP/XML server are similar to the Remote Log Server (Figure 6.5). Additionally, a stylesheet can be specified (Figure 6.8). A reference to the stylesheet will then be added to the XML output. The browser will request the stylesheet from the *CollectionConnection* server. The stylesheet can contain transformation instructions but the XML browser that is used must be able to process them.

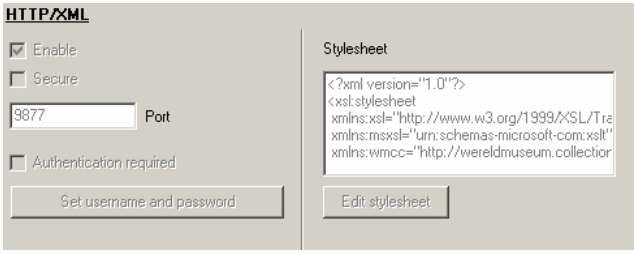


Figure 6.8. HTTP/XML settings

There are two related parameters for stylesheets:

- `ignorestylesheet=yes`. If this parameter is used, the default stylesheet will be ignored.
- `Stylesheet=location`. This wil override the default stylesheet and add a stylesheet reference to `location` in the returned XML document, for example:

```
http://localhost:9877?action=get&command=s
earch&query=*=*&range=1-
10&fields=*&stylesheet=http://localhost:98
79/cc.xsl
```

6.2.9 Soap

Soap provides an easy way to integrate *CollectionConnection* in modern programming environments. The Soap protocol provides one Interface:

`IcollectionConnectionSoapServer`.

The WSDL for the inteface can be retrieved in the following way:

`http://server:port/wsdl/ICollectionConnectionSoapServer`.

The interface contains one message ("`getObjectSet`") with the following parameters:

```
<part name="command" type="xs:string" />
<part name="parameters" type="xs:string" />
<part name="minRow" type="xs:int" />
<part name="maxRow" type="xs:int" />
<part name="sort" type="xs:string" />
<part name="predict" type="xs:string" />
<part name="dcl" type="xs:string" />
<part name="username" type="xs:string" />
<part name="password" type="xs:string" />
```

The meaning of the parameters is discussed in Section 6.3.6 on page 103. The settings of the Soap server are similar to the Remote Log Server port (Figure 6.5).

6.2.10 HTTP/HTML-ASP

The HTTP/HTML-ASP component provides a Microsoft IIS ASP compatible webserver. The webserver component in the *CollectionConnection* server can handle ASP files, but has some limitations compared to IIS and PWS:

- ActiveServer cannot mix more scripting languages. In IIS/PWS, one ASP page can 'mix' scripting languages. This option is not available in *CollectionConnection*. Only one scripting language can be used for server-side scripting.
- The ActiveServer Component is not binary compatible with IIS/PWS. This difference is obvious when making IIS extension objects using Visual Basic or other tools. The ActiveServer Component implements all objects, methods and properties of the IIS/PWS, but the interfaces are not binary compatible.
- The ActiveServer component is not directly compatible with WEB applications / DLLs compiled by the VB6 IIS Application. Due to the special mechanism how VB6 deploys and works with the IIS, these DLLs cannot be used (without 'twisting') with *CollectionConnection*.
- Response.Transfer and Response.Exec functions. These functions are defined in the ActiveServer component interface, but they are not implemented.
- Response.Flush will stop the output of the program. Due to internal functioning, when doing Response.Flush the component will flush the output and finish the stream to the browser.
- The component is always using 'Buffer On' option. The buffer is always on, regardless of the Response.Buffer directive
- *CollectionConnection* will not process ISAPI/CGI extensions.
- *CollectionConnection* does not implement Session and Application events. Events (Session_OnStart, Session_OnEnd, Application_OnStart, Application_OnEnd) are not supported.
- *CollectionConnection* does not call the OnStartPage() ActiveX method.
- Locale settings are user-defined. The locale that is used for executing ASP pages is the user-defined locale. On the contrary, IIS/PWS uses the installed system locale.
- Session objects are threadunsafe. *CollectionConnection* does not include any protection for the objects stored in the session variables (no multithreading protection).
- Application objects are threadsafe.

To use the integrated webserver, a computer must have installed the Windows Scripting Host (WSH) and a Winsock 2.0 library.

General settings
Figure 6.9 shows the general settings for the HTTP/HTML-ASP server.

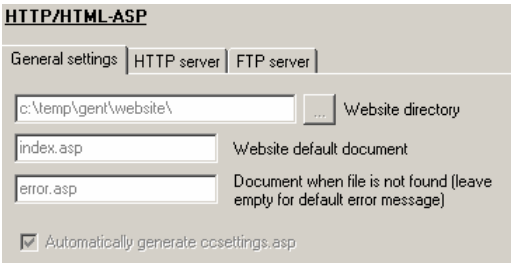


Figure 6.9. HTTP/HTML-ASP general settings

- *Website directory*. The webserver will treat this directory as its root. If this is left empty, it will take the directory of the path where the executable is stored. The website directory can be specified as a relative path, and it will then be relative to the path of the executable, for example, "`website`", "`..\website`" or "`\website`".
- *Website default document*. This document will be loaded when a webbrowser does not request a document specifically.
- *Error document*. This document will be loaded when the requested document is not found.
- Automatically generate `ccsettings.asp`. When checked, a file with the name 'ccsettings.asp' will be dynamically created by the server when it is requested. If the file already exists, it will be overwritten! An example is shown in Figure 6.10. The variables will be set with the current values from the server. This functionality can be used by including this file in other asp files with `<!--#include file="ccsettings.asp"-->`, and then setting the client object properties with these variables. Note that for authentication, you will need to include the password yourself. For security reasons, the server does not know or store the passwords that are configured, and the appropriate password can therfore not be be included in `ccsettings.asp` automatically.

```
<%
   globalCCPort="9876"
   globalCCServer="127.0.0.1"
   globalSSL=True
   globalLicensePath="D:\data\ccServer\demo"
   globalccusername="demo-user"
%>
```

Figure 6.10. Example 'ccsettings.asp'

HTTP server settings

The settings of the HTTP/XML server (see Figure 6.11) are similar to the Remote Log Server (Figure 6.5).

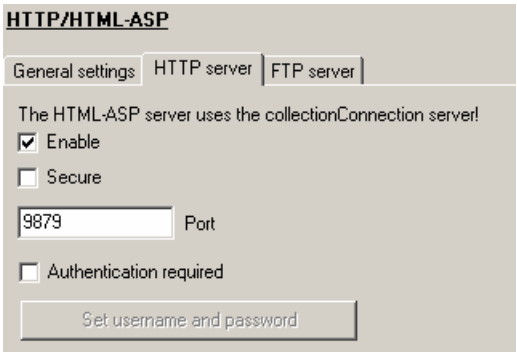


Figure 6.11. HTTP/HTML-ASP HTTP server settings

If secure is enabled, communication takes place by SSL. Note that, due to the ASP functionality, the SSL implementation differs slightly from the other servers. If enabled, a secured server is activated on the specified port. Another unsecured server is opened on a port in the range 59000-60000. This unsecured server is bound to `localhost`., i.e., it can not be reached by other computers. The requests to the secured server are forwarded to the unsecure server.

FTP server settings
The FTP server settings are shown in Figure 6.12. The server can be connected to by a normal FTP-client. The files are stored in the website directory configured in the HTTP/HTML-ASP server (see also Figure 6.9). The FTP server needs a username and a password.

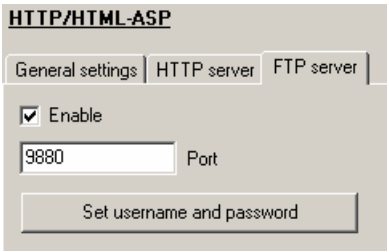


Figure 6.12. HTTP/HTML-ASP FTP server settings

6.2.11 Open Archives Initiative (OAI)

CollectionConnection fully supports version 2.0 of the OAI-protocol. The settings of the OAI protocol are similar to the Remote Log Server (Figure 6.5). One addition is the setting 'Strict OAI' (Figure 6.13). When this is enabled, the XML-records in the resultset are supposed to conform to the Dublin Core format. Tags that are not part of this format will be removed. When this is disabled, all tags are allowed but then the OAI protocol is not strictly followed. An example of an OAI request is:

```
http://localhost:9881?verb=listrecords&metadataprefix=oa
i_dc
```

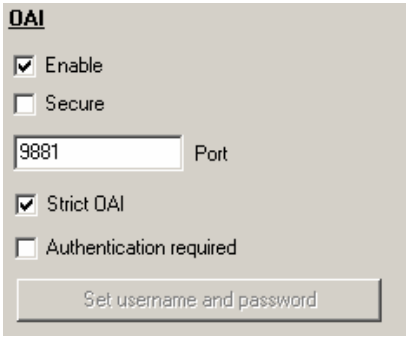


Figure 6.13. OAI server settings

Please refer to `http://www.openarchives.org` for a full description of the Open Archives Initiative.

6.2.12 Image server

The image server is a special purpose HTTP server. It not only serves images, it can also perform server-side manipulations on the image such as sizing and adding a watermark. For this, it uses a special file format with the extension .cci for *CollectionConnection* Image. Chapter 11 contains an elaborate description of the functionalities of the image server. In this chapter, only the settings that are configured in the *CollectionConnection* server are discussed.

General settings
There are three general settings for the image server (see Figure 6.14):

- The directory where the images are stored.
- The quality of the images that are created dynamically. Higher quality means a larger file and a higher processing time.
- The maximum width and maximum height that the image may have, specified in pixels. *CollectionConnection* can handle images with a very high resolution, but it might be undesirable to publish the source images with such a high quality. Therefore, the width and height can be restricted.

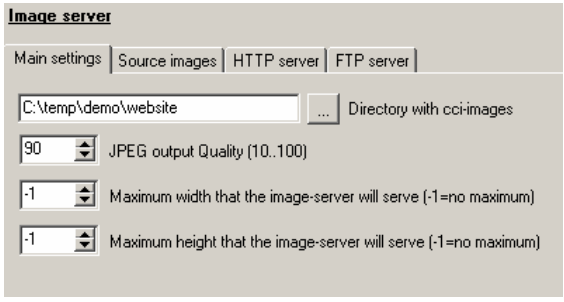


Figure 6.14. Image server general settings

Source images

JPEG Images can be converted in Windows Explorer to CCI-images (see Chapter 11), but they can also be converted by the *CollectionConnection* server. When a JPEG-image is requested from the image-server, it first checks whether the images is already converted. If not, or if the source image has been changed, it will convert the source image with the given parameters (see Figure 6.15). Then, it will generate the requested image (with the requested width, height, quality, border settings, etc.) from the CCI-Image. If the CCI-image already exists, no conversion will take place and the Image server will immediately create and serve the requested image.

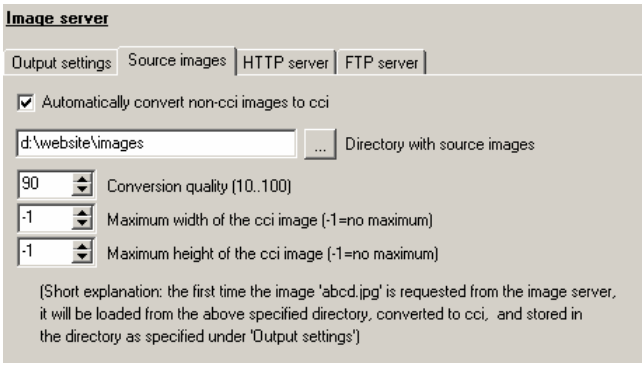


Image server
Output settings
Source images
HTTP server
FTP server
Automatically convert non-cci images to cci
d:\website\images
...
Directory with source images
90
Conversion quality (10..100)
-1
Maximum width of the cci image (-1=no maximum)
-1
Maximum height of the cci image (-1=no maximum)
(Short explanation: the first time the image 'abcd.jpg' is requested from the image server, it will be loaded from the above specified directory, converted to cci, and stored in the directory as specified under 'Output settings')

Figure 6.15. Image server automatic conversion

HTTP server settings

The settings of the HTTP/Image server (see Figure 6.16) are similar to the Remote Log Server (Figure 6.5). There is one extension. To prevent a broken image icon in the webbrowser, a default image can be specified that will be served when the requested image does not exist.

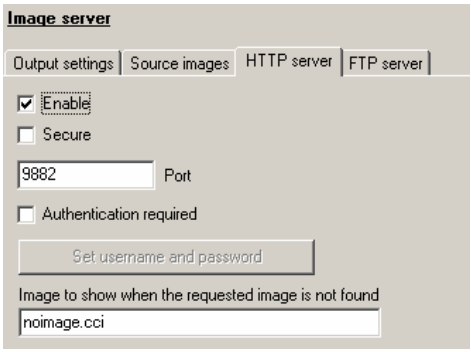


Figure 6.16. Image server HTTP server settings

FTP-server settings

Images can be uploaded to the server using the FTP-server. The FTP-server settings are shown in Figure 6.17. The server can be connected to by a normal FTP-client. The files

are stored in the image directory configured in the general settings. The FTP server needs a username and a password.

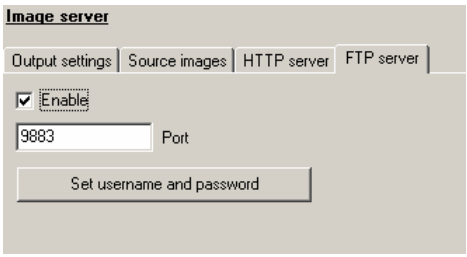


Figure 6.17. Image server ftp server settings

6.2.13 Web crawler gateway

The web crawler gateway provides straightforward access to the data in the index. Each record will be served as a HTML file. The web crawler gateway provides an efficient way to let search engines such as Google and Alta Vista include the information from the index. Search engine spiders usually have a maximum number of links on a page that they index and a maximum depth of links on a site. To maximize the number of objects that will be visited by such spiders, the web crawler gateway in *CollectionConnection* provides a hierarchy of links to all objects in the collection, thereby limiting both the number of links on one page and the depth of links.

General settings

The following general settings must be specified (Figure 6.18):

- The tag that uniquely identifies the objects. Note that the tag is case sensitive!
- A script that converts XML to HTML. For an example, see Figure 6.20. This example is generated with the following general purpose script that converts an XML-record is to HTML with an XSL transformation:

```
Dim myXML,myXSL,xsl

xsl="<?xml version='1.0'?>"
xsl=xsl+"<xsl:stylesheet version='1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>"
xsl=xsl+"<xsl:template match='/'>"
xsl=xsl+"<HTML>"
xsl=xsl+"  <center><P>Welcome to the CollectionConnection
  Demo Crawler Gateway</P></center>"
xsl=xsl+"     <table width='100%' border='1'
  cellpadding='2' cellspacing='0'>"
xsl=xsl+"       <tr>"
xsl=xsl+"         <td><b>Publisher:</b></td>"
xsl=xsl+"         <td><xsl:value-of
  select='collectionConnection-
  resultset/publisher'/></td>"
xsl=xsl+"       </tr>"
xsl=xsl+"       <tr>"
xsl=xsl+"         <td><b>Identifier:</b></td>"
```

```
xsl=xsl+"         <td><xsl:value-of
  select='collectionConnection-resultset/dc-
  record/identifier'/></td>"
xsl=xsl+"       </tr>"
xsl=xsl+"       <tr>"
xsl=xsl+"         <td><b>Title:</b></td>"
xsl=xsl+"         <td><xsl:value-of
  select='collectionConnection-resultset/dc-
  record/title'/></td>"
xsl=xsl+"       </tr>"
xsl=xsl+"       <tr>"
xsl=xsl+"         <td><b>Description:</b></td>"
xsl=xsl+"         <td><xsl:value-of
  select='collectionConnection-resultset/dc-
  record/description'/></td>"
xsl=xsl+"       </tr>"
xsl=xsl+"       <tr>"
xsl=xsl+"         <td><b>Website URL:</b></td>"
xsl=xsl+"         <xsl:variable name='lnk'
  select='string(collectionConnection-resultset/dc-
  record/identifier)'/>"
xsl=xsl+"         <xsl:variable name='lnk2'
  select='concat("+chr(34)+"http://localhost:9879/basicsea
  rch.asp?field1=*&amp;operator1=all&amp;sort=ccrelevance&
  amp;showtype=single%20objects&amp;searchfor1="+chr(34)+"
  ,$lnk)'/>"
xsl=xsl+"         <td><a href='{$lnk2}'>Click here to go
  to this object in our website</a></td>"
xsl=xsl+"       </tr>"
xsl=xsl+"    </table>"
xsl=xsl+"    <br/>"
xsl=xsl+"</HTML>"
xsl=xsl+"</xsl:template>"
xsl=xsl+"</xsl:stylesheet>"

myXML=CreateOleObject("Microsoft.XMLDOM")
myXSL=CreateOleObject("Microsoft.XMLDOM")
myXML.async=False
myXSL.async=False
myXML.loadXML(ccXMLRecord)
myXSL.loadXML(xsl)
ccXMLRecord=myXML.transformNode(myXSL)
```

- A HTML template. The links to the individual records will be put in this template before the `<body>` tag. See Figure 6.19.

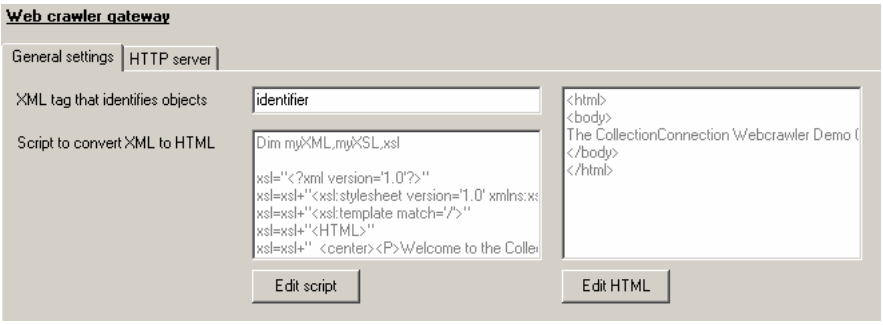


Figure 6.18. Web crawler gateway general settings

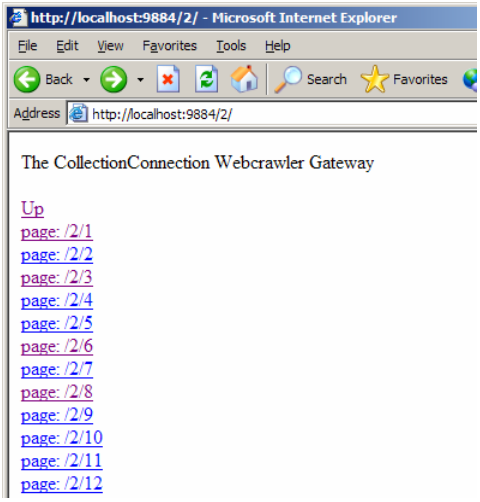


Figure 6.19. Web crawler gateway page with links

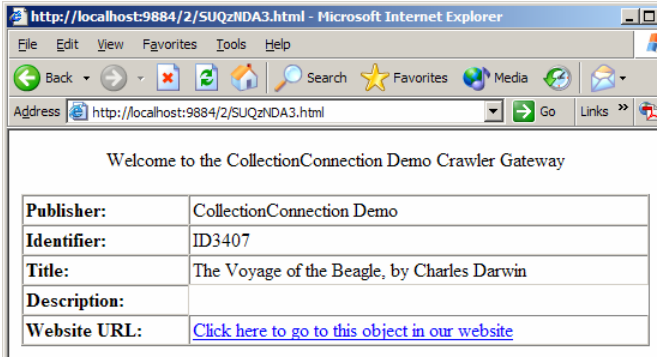

http://localhost:9884/2/SUQzNDA3.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9884/2/SUQzNDA3.html
Go Links
Welcome to the CollectionConnection Demo Crawler Gateway
Publisher: CollectionConnection Demo
Identifier: ID3407
Title: The Voyage of the Beagle, by Charles Darwin
Description:
Website URL: Click here to go to this object in our website

Figure 6.20. Web crawler gateway page with object

HTTP server settings

The settings of the web crawler gateway (see Figure 6.21) are similar to the Remote Log Server (Figure 6.5).

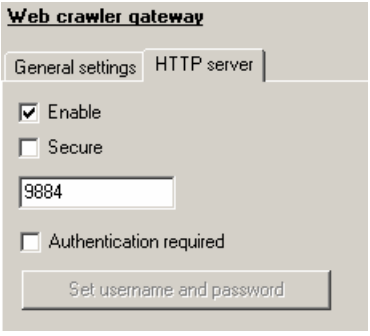


Figure 6.21. Web crawler gateway HTTP server settings

6.2.14 Z39.50

The Z39.50 server can be used to open the collection for Z39.50 clients such as Bookwhere. The server can not be secured with SSL and it can not use authentication. Three settings are available. First, the port needs to be specified (Figure 6.22).

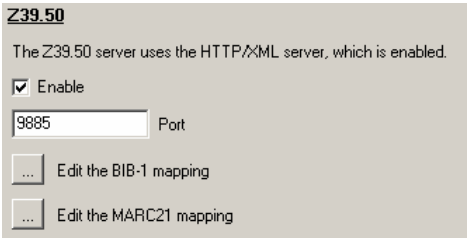


Figure 6.22. Z39.50 server settings

Second, the mapping of BIB1 use attributes to XML tags must be specified (Figure 6.23). This is used to map requests from Z39.50 clients to a *CollectionConnection* query.

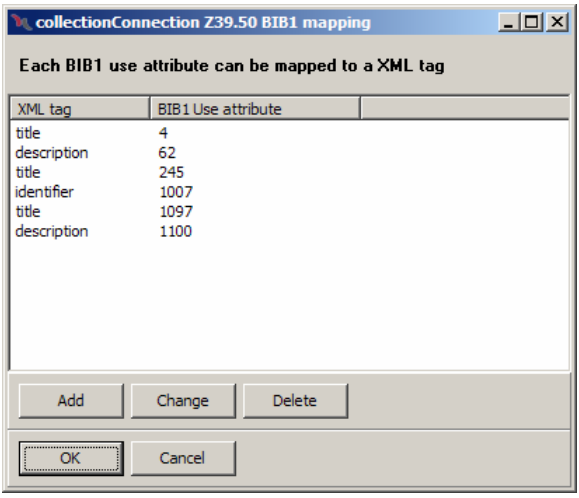


Figure 6.23. Z39.50 - BIB1 Use attribute to XML tag

Third, the XML structure must be mapped to a MARC21 structure (Figure 6.24). The MARC21 records are used by the Z39.50 client to show the results of the query.

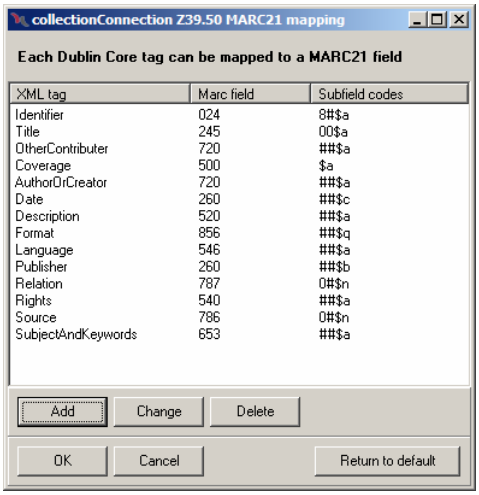


Figure 6.24. Z39.50 - XML tag to Marc field

For more information about Z39.50, see `http://www.loc.gov/z3950/agency/`.

6.2.15 Index transmission

The index must be in the same directory as the server profile. If a profile is opened, however, the corresponding index is locked and can not be replaced by the operating system. *CollectionConnection* provides four ways to update an index when the profile is opened.

Index transmission by FTP

The index can be sent by a normal FTP client. The index will only be updated after the FTP client closes the connection! A port, username, and password need to be specified (Figure 6.25).

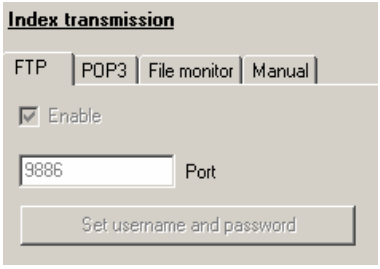


Figure 6.25. Index transmission by ftp

Index transmission by Email

The index can be mailed by a normal Email client. The server needs to be configured to check the Email address periodically (Figure 6.26). If it finds an Email that has an appropriate index attached, it will download the email, extract the attachment, and replace the current index with it.

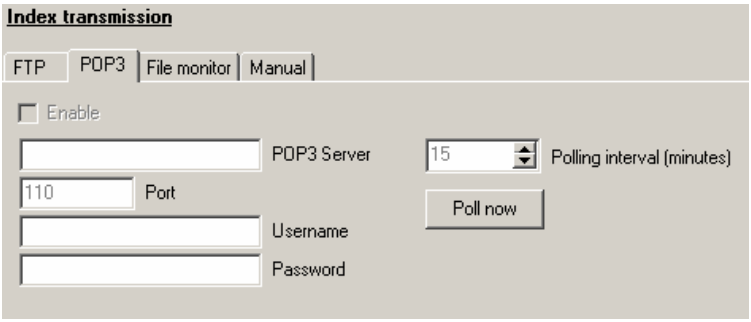


Figure 6.26. Index transmission by mail

Index transmission by monitoring a directory
CollectionConnection can monitor a directory and, when an index is put there, replace the index with a new one. This can be used to let a user upload an index via Internet Explorer, and let *CollectionConnection* monitor the upload directory.

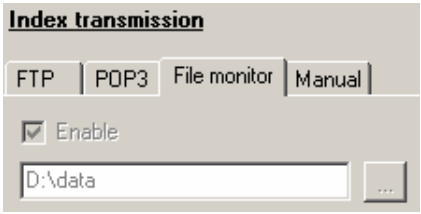


Figure 6.27. Automatic index transmission

Manual index transmission
An index can be loaded manually. The file that is chosen needs to be a valid index for the current server profile.

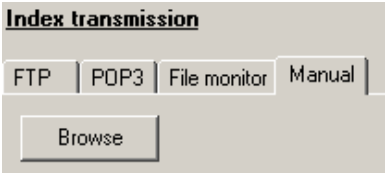


Figure 6.28. Manual index transmission

In all cases, *CollectionConnection* will check the integrity of the new index. If it is not valid, the current index will not be replaced.

6.3 Query parameters

The *CollectionConnection* server knows commands to retrieve the following information:

- Collections
- Thesaurus
- Spelling alternatives

- Index data structure
- Full text searching

Each of these information categories has one or more commands that will be discussed in the following sections. The *CollectionConnection* Client (see chapter 8) uses these command to communicate with the server. Note that it is not necessary to know the underlying commands when the *CollectionConnection* Client is used.

6.3.1 Digital signatures

Some of the server commands return records with object information. Such records can be digitally signed. With the signature, the receiver of the information can check whether the information has been changed after it has been created.

Upon request, a signature is added to the record. The signature contains the name of the license holder, a public key, and a signature value. The signature value is a digest of the original record that is encrypted with the private key that is assigned to the license holder. The signature can be verified with the public key that is contained in the signed record. If there is the suspicion that both public key and signature value are forged, the receiver of the information can request the public key from the license holder to check. The license holder can publish the public key safely on an SSL protected website where receivers of records can obtain it. The following example shows a signed record:

```
<dc-record>
  <publisher>CollectionConnection Demo</publisher>
  <identifier>ID01</identifier>
  <title test="1" abc="b928">atland, by Edwin A. Abbott</title>
  <filename>flatxxx.xxx1</filename>
  <test>b928</test>
  <sortKeyValue>ID01</sortKeyValue>
  <recordnumber>4</recordnumber>
   <Signature>
      <licensee>CollectionConnection Demo</licensee>
      <publicKey>AKk1TtG17DIJwrXzT3xpLf+mw01BK2G
          5Ivzp7mqsp+INf0Rz0kBqm1zdDKEJsTfJru52wwf
          mvZ0dRMCkipkUmZ6RaCP+1xvPzYJxdFGykZY2Vh
          Nzc12UvjeOCAGnmGFuKdyv0jGWGLB/JW0uN2m
          sMeO2je8scy9n1NnIw2r0VjcjA0AB
      </publicKey>
      <SignatureValue>k2WHpEpilGBWnLmh3ebzBxN25E
          R+tJ4gCP+zqWCKa7mq9X/No8WFS3e3rUa033L+
          W5XmSVsrHpd3M058HXgkWGXd9xqxgWlCk87M4
          2c70ujNo+DjMaz+ChpFBywE7mpdEAfVyxxWAPcV
          8Ylof+fk09kwOCvdrIwAhSAeieWKE98=
      </SignatureValue>
    </Signature>
</dc-record>
```

The *CollectionConnection* client provides a function to verify a signed message, see section 8.10 on page 128. This function is used in an example ASP file, see Chapter 9.

6.3.2 Collections

Objects can be grouped in collections. The nature of a collection is not relevant for *CollectionConnection*. They can be objects that are acquired in the same year, objects that are stored in the same room, objects that are from the same era, etc. It is possible to browse through the collections, to see what collections an object belongs to, and to see

what collections an object is in. Furthermore, collections can be grouped themselves. A collection can only be a member of one group.

There are three collection related commands:

- `Collectiongrouplist`. This provides a list with groups of collections. The following parameter is required:
  a. `Range`. The start and end record to retrieve, e.g., 1-10.

- `Collectionlist`. Provides a list with available collections. There are three parameters:
  a. `Query`. Optional. Specifies the collectiongroup.
  b. `Range`. The start and end record to retrieve, e.g., 1-10.
  c. `Fields`. A comma seperated list with the required fields or tags. Possible fields (see the collection list query in Table 5.3) are:
     `Identifier`.
     `Title`.
     `Group`.
     `Description`.
     `Count`. The number of objects in the collection.

- `Objectcollection`. Provides the objects that belong to a collection using the following parameters:
  a. `Query`. The collection identifier.
  b. `Range`. The start and end record to retrieve, e.g., 1-10.
  c. `Fields`. A comma seperated list with the required fields or tags. Use a '*' to get all fields.
  d. `Sort`. The field on which the result-set should be sorted.
  e. `Digitalsignature`. Specifies whether a digital signature should be added. Specify '`digitalsignature=yes`' to enable this.
  f. `DCL`. Specifies the Dynamic Content Links that should be added to the resultset (see the description of Dynamic Content Links in section 5.6.13).

Examples (the examples use the HTTP/XML server, see 6.2.8):

- `http://localhost:9877/action=get&command=collectiongroup list&range=1-100`

- `http://localhost:9877/action=get&command=collectionlist& range=1-15&fields=identifier,title,description,count`

- `http://localhost:9877/action=get&command=objectcollectio n&query=1&range=1-10&fields=*& sort=identifier& dcl=thesaurus`

6.3.3 Thesaurus

CollectionConnection has support for a hierarchical structure of concepts or terms, commonly referred to as a thesaurus. Users can browse through the thesaurus itself,

they can see what objects belong to an element in the thesaurus, and they can see what concepts in the thesaurus the object belongs to. This makes it possible for users to find related objects in a structured way.

- `Categorychildren`. Provides a list with nodes that fall directly under a thesaurusnode. Each record in the resultset specifies the number of objects that are categorized on or under the node, and the number of direct children of the node itself.
  a. `Query`. The thesaurus node. Leave empty for the root nodes.
  b. `Range`. The start and end record to retrieve, e.g., 1-10.

- `Categoryobjects`. Provides a list with objects that belong to a thesaurusnode.
  a. `Query`. The thesaurus node.
  b. `Range`. The start and end record to retrieve, e.g., 1-10
  c. `Fields`. A comma seperated list with the required fields or tags. Use a '*' to get all fields.
  d. `Sort`. The field on which the result-set should be sorted.
  e. `Digitalsignature`. Specifies whether a digital signature should be added. Specify '`digitalsignature=yes`' to enable this.
  f. `DCL`. Specifies the Dynamic Content Links that should be added to the resultset (see the description of Dynamic Content Links in section 5.6.13 on page 74).

Examples:

- `http://localhost:9877/action=get&command=categorychildren&query=0VM.AAA&range=1-10`

- `http://localhost:9877/action=get&command=categoryobjects&query=0VM.AAA.AAB&range=1-10&fields=*&sort=identifier`

6.3.4 Spelling alternatives

CollectionConnection can provide spelling alternatives for a word. These alternatives are words that occur in the index and that look or sound like the specified word. The list with alternatives is sorted on similarity. There is one command with one parameter:

- `Spell`
  a. `Word`. The word for which spelling alternatives are sought.

Example: `http://localhost:9877/action=get&command=spell&word=boeda`

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <collectionConnection-resultset>
  <publisher>CollectionConnection Demo</publisher>
  <type>search</type>
  <type>cultural</type>
  <title>spelling suggestion</title>
- <word>
  boeda
     <alternative score="91">boedai</alternative>
```

```
<alternative score="91">boedah</alternative>
<alternative score="91">boedha</alternative>
<alternative score="88">boedak</alternative>
<alternative score="88">boedaq</alternative>
<alternative score="87">boda</alternative>
<alternative score="82">boua</alternative>
<alternative score="82">boeta</alternative>
<alternative score="82">boeka</alternative>
<alternative score="82">oeda</alternative>
<alternative score="81">boeah</alternative>
<alternative score="81">boeat</alternative>
<alternative score="80">boeddha</alternative>
....
```

This functionality is especially useful when a search request has resulted in zero or very few results. Chances are that the user spelled the requested word wrongly. The *CollctionConnection* Client contains functions to retreive a number of spelling alternatives using this command.

6.3.5 Index data structure

Each record or object in the index is an XML structure with a number of tags. The tags and their characteristics are determined by the *CollectionConnection* Indexer. With the 'tags' command, all tags are returned:

- `Tags`. This command returns a list with tags that are specified in the indexer. There are no parameters. Each tag in the resultset specifies the properties of a tag in the object structure.

Example: `http://localhost:9877/action=get&command=tags`

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <collectionConnection-resultset>
  <publisher>CollectionConnection Demo</publisher>
  <type>search</type>
  <type>cultural</type>
  <title>list with tags</title>
  <count>12</count>
- <tag>
    <name>identifier</name>
    <datatype>string</datatype>
    <parsed>no</parsed>
    <indexed>yes</indexed>
    <sorted>yes</sorted>
    <browseable>yes</browseable>
  </tag>
- <tag>
    <name>description</name>
    <datatype>string</datatype>
    <parsed>yes</parsed>
    <indexed>yes</indexed>
    <sorted>yes</sorted>
    <browseable>no</browseable>
    <attribute>type="scientific"</attribute>
    <attribute>type="public"</attribute>
  </tag>
....
```

- `Browsetag`. This command returns a list with values that exist for a given tag. This only works for thags that have this option enabled in the indexer (see section 5.6.11.3 on page 71). This is specified by the `<browseable>` tag as can be seen above.
  b. `Tag`. This is the tag that needs to be browsed.
  c. `Range`. The start and end record to retrieve, e.g., 1-10.

Example:

```
http://localhost:9877/action=get&command=browsetag&tag=identifier&rang
e=11-20
```

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <collectionConnection-resultset>
    <publisher>CollectionConnection Demo</publisher>
    <type>browse</type>
    <type>cultural</type>
    <title>list with fieldvalues for tag identifier</title>
    <count>235243</count>
    <firstRequested>11</firstRequested>
    <lastRequested>20</lastRequested>
  - <dc-record>
      <publisher>CollectionConnection Demo</publisher>
      <type>original</type>
      <type>cultural</type>
      <title>00-1017</title>
    </dc-record>
....
```

6.3.6 Full text search

The full text search command searches in text for the specified search words or numbers. Full text search has the following functions (see also section 5.6.11.3):

- Searching for words
- Postfix, suffix, and substring search for words
- Searching for phrases
- Searching for complete field values
- Searching for numbers or dates
- Searching for ranges of numbers or dates
- Combining queries
- Combining queries with thesaurus node membership or collection membership

The following parameters can be used when constructing queries:

- `Query`. The query specification. The specification formats will be described under the parameter specification.
- `Range`. The start and end record to retrieve, e.g., 1-10
- `Fields`. A comma seperated list with the required fields or tags. Use a '*' to get all fields.
- `Sort`. The field on which the result-set should be sorted. If it is not specified, the results are sorted on the field "identifier". The specification of the sort field must be preceded by "sort=", for example: "sort=title". When sort is specified as "ccRelevance", the

results are sorted on proximity and frequency of the specified search words.

- `Digitalsignature`. Specifies whether a digital signature should be added. Specify 'digitalsignature=yes' to enable this.
- `DCL`. Specifies the Dynamic Content Links that should be added to the resultset (see the description of Dynamic Content Links in section 5.6.13 on page 74).
- `Predict`. This parameter specifies one or more additional queries. CollectionConnection will combine each additional query with the search query and calculate the size of the resultset. The additional queries must be URLEncoded and separated by a colon (:). Then, the resulting string must be URLEncoded again.

  Example:

  ```
  predict=inthes%2528IC0%252EAAJ%252EAAA%2529%3Ainthes
  %2528IC0%252EAAJ%252EAAB%2529%3A
  ```

  Is the predict specification for:

  ```
  Inthes(IC0.AAJ.AAA)
  Inthes(IC0.AAJ.AAB)
  ```

  This will append the following two tags to the resultset:

  ```
  <predict
  query="inthes%28IC0%2EAAJ%2EAAA%29">9293</predict
  >
  <predict
  query="inthes%28IC0%2EAAJ%2EAAB%29">8231</predict
  >
  ```

  9293 will be the size of the resultset when `inthes(IC0.AAJ.AAA)` is combined with the original query. 8231 will be the size when the original query is combined with `inthes(IC0.AAJ.AAA)`.

CollectionConnection can extract parts of tags where search words occur. This can be used to summarize the search results with large text fields. The context of the first few occurrences of the word in the text is shown, the rest of the text is omitted. With queries that consist of multiple words, the summary list will contain the context of all words evenly. The following parameters can be added to the command string:

- `summaryThreshold`. This specifies the minimum number of characters that a tag must have before it is summarized.
- `maxSummaryLines`. The maximum number of pieces of text that will be in the summary.
- `optimalSummaryLineLength`. The length of one piece of summarized text. *CollectionConnection* will try to find sentences of this size.
- `maxSummaryLineLength`. If a sentence gets very long, *CollectionConnection* will cut it off at the maximum line length.
- `summaryLineStartText`. The text that is put before each text extract.
- `summaryLineEndText`. The text that is put after each text extract.
- `summaryFields`. The tagnames of tags that should be summarized. This is a comma separated list. Leave it empty to have all fields summarized.

The example in Figure 6.29 shows the results of searching for 'flower' and summarizing the description field.

Parameters:
```
summaryThreshold=75&maxSummaryLineLength=50&optimalSummaryLineLe
ngth=30&maxSummaryLines=5&summaryLineStartText=...&summaryLineEn
dText=...%3Cbr%3E&summaryFields=description
```

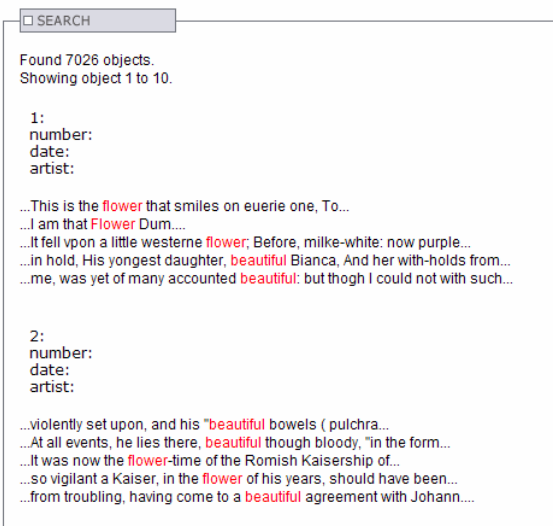

SEARCH
Found 7026 objects.
Showing object 1 to 10.
1:
number:
date:
artist:
...This is the flower that smiles on euerie one, To...
...I am that Flower Dum....
...It fell vpon a little westerne flower; Before, milke-white: now purple...
...in hold, His yongest daughter, beautiful Bianca, And her with-holds from...
...me, was yet of many accounted beautiful: but thogh I could not with such...
2:
number:
date:
artist:
...violently set upon, and his "beautiful bowels (pulchra...
...At all events, he lies there, beautiful though bloody, "in the form...
...It was now the flower-time of the Romish Kaisership of...
...so vigilant a Kaiser, in the flower of his years, should have been...
...from troubling, having come to a beautiful agreement with Johann....

Figure 6.29. Result of summarizing

Query specification
Queries are specified with the triple `<field><operator><value>`. Queries can be nested with negation ("not"), conjunction ("and"), and disjunction ("or").

Each of the search query parameters will be explained below.

Field
Searching can be either performed on all fields or limited to one or more fields.

- To search in all fields, use * as the field specifier
- To search in multiple fields, separate the fieldnames by a comma, e.g.: description,title

- Special field specifiers:
  a. `ccImplicitThesaurus`. Adds thesaurus nodes of which the node term was found in one of the tags of the XML-record.
  b. `ccIndexName`. The name of a combined index.

Operator
Operators for strings are different from operators for numbers:

- Operator for strings

`=` : search words occur somewhere in the specified fields

- Operators for numbers and dates:
  `=` : equal to
  `>` : larger than
  `<` : smaller than
  `>=` : larger than or equal to
  `<=` : smaller than or equal to

Value
As with operators, the value specification differs with the different data types.

Value specification for strings:

- Simple word search: specify the word.
  Example: `description=test`

- Simple word search with wild cards.
  Example postfix: `test`*
  Example suffix: *`tes`
  Example substring search: *`tes`*

- Advanced word search: specify multiple words. When sorting is specified as '`ccRelevance`' the resultset will be ordered on proximity and frequency of the search words in the records.
  Example: `description=this is a test`

- Phrase search: enclose multiple words with double quotes.
  Example: `description="this is a test"`

- Exact field value search: enclose the word or words with square brackets (the tag must have enabled the 'browse through all values' setting in the tag settings window of the Indexer).
  - Example: `description=[this is a test]`

Value specification for numbers:

- The number without a thousands separator and a period (.) as the fraction separator.
  Example: `number=12332`
  Example: `number=322.123`

Value specification for dates:

- Dates should be specified as dd-mm-yyyy hh:mm:ss
  Example: `created=`05-02-2004 12:33:21

Negation
Negation executes a query and returns the records that do not conform to the query.

- `Not(query)`
  Example: `Not(description=test)`

Conjunction
Conjunction takes two queries and returns the records that conform to both queries.

- `and(query1;query2)`
  Example: `and(description=test;title=water)`

Disjunction
Disjunction takes two queries and returns the records that conform to either one.

- `or(query1;query2)`
  Example: `or(description=test;title=water)`

Nesting
Queries can be nested, for example:

- `and(title,description=painting; yearOfcreation<1800)`
- `or(all=paint*; inthes(aat.d))`
- `and(or(and(all=paint*;not(all=color));all=snow);yearOfCr eation>1900)`

Thesaurus node membership
Thesaurus membership can be tested with the "`inthes`" search command. This is equivalent to the Categoryobjects command. The resultset will contain the records that are classified on or under the specified thesaurus node. The inthes command can be mixed with other search commands.

- `inthes(thesaurus node)`
  Example: `and(inthes(aat.vbo.abc);title=snow)`

Collection membership
Collection membership can be tested with the "`incollection`" search command. The "`incollection`" search command is equivalent to the `Objectcollection` command. The resultset will contain the records that are in the specified collection. The `incollection` command can be nested with other search commands.

- `incollection(collection identifier)`
  Example: `and(incollection(123);title=snow)`

6.3.7 Authentication
When the commands are sent with the *CollectionConnection* protocol, authentication information must be added to the query: `username=username&password=password`

- Example:
  `action=get&command=browsetag&tag=identifier&range=11-20&username=demouser&password=demopassword`

With the HTTP/XML protocol, HTTP-authentication is used, and the username and password do not have to be appended do the query.

6.3.8 Distrubuted search

All commands can also be used in distributed search (see Chapter 7). All commands and parameters stay the same, with one exception: the "range". This parameter is replaced with three other parameters:

- `State`: the state of the previous query (see Chapter 7)
- `Direction`: forward or backward
- `Num`: the number of records to retrieve

The Active Server Pages (that are part of the *CollectionConnection* system; see Chapter 9) in combination with the *CollectionConnection* Client provide extensive functionality to stepwise build search queries with the available commands.

6.3.9 Output

The output of a query is always XML. XML-Records are put in a `<CollectionConnection-resultset>`. This root record specifies the query that has delivered the record set, what objects from the result set are requested, how many records conform to the query, how the records are sorted, and the licensee. For example:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <collectionConnection-resultset>
  <licensee>CollectionConnection Trial</licensee>
  <publisher>CollectionConnection Demo</publisher>
  <type>search</type>
  <type>cultural</type>
  <title>search *=*</title>
  <count>103</count>
  <firstRequested>1</firstRequested>
  <lastRequested>10</lastRequested>
  <sortKey>identifier</sortKey>
- <dc-record>
  <publisher>CollectionConnection Demo</publisher>
      <0therContributer>Museum     voor     Schone     Kunsten
  (Gent)</0therContributer>
  <identifier>1820-C</identifier>
   <title>De schone Anthia begeeft zich aan het hoofd van haar
  gezellinnen naar de tempel van Diana te Ephesus</title>
   <authororcreatorsort>Paelinck, Joseph</authororcreatorsort>
   <medium>olieverf op doek</medium>
   <objectname>schilderij</objectname>
    <Description>In  De  schone  Anthia  brengt  Joseph  Paelinck
  (1781-1839) een episode uit een 0udgrieks verhaal in beeld.
  </Description>
   <format>hoogte 233 cm , breedte 300 cm</format>
   <image>..\\images\S-207.jpg</image>
   <period>19de eeuw</period>
   <movement>neoclassicisme</movement>
   <title>La belle Anthia marchant à la tête de ses compagnes au
  temple de Diane à Ephèse</title>
    <title>The Fair Anthia Leading her Companions to the Temple
  of Diana in Ephesus</title>
     <title>Die  schöne  Anthia  führt  ihre  Gefährten  zum
  Artemistempel in Ephesos</title>
   <begindate>1820</begindate>
   <authororcreator>Joseph Paelinck</authororcreator>
```

```
<sortKeyValue>1820-C</sortKeyValue>
<recordnumber>1</recordnumber>
</dc-record>
```

The following meta-information is added to each record by the *CollectionConnection* Server:

- `Publisher`: the publisher of the record as specified in the server settings.
- `sortKeyValue`: The value on the basis of which the place of the record is determined.
- `Recordnumber`: the number of the record in the record-set. This field is important because the order of tags in XML may not signify any meaning.

As specified, the resultset can also contain `Predict` tags when requested. Furthermore, the records can contain `DCL` tags when requested.

Binary contents is Mime-encoded. Because the size of binary contents may be large, Mime-encoded binary contents are not stored in the XML-records themselves. By reading binary tags only when explicitly requested, performance is preserved. Binary tags in XML-records contain an identifier that can be used to request the actual (mime-encoded) binary contents:

```
<testimage>attachment:4946</testimage>
```

The mime-encoded data can be retrieved in two ways. First, the 'attachment'-command can be used to get the data of one attachment:

- `http://localhost:9877/action=get&command=attachment&query=attachment:4946`

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <collectionConnection-resultset>
  <publisher>CollectionConnection Demo</publisher>
  <type>attachment</type>
  <type>cultural</type>
  <title>attachmentid=attachment:4946</title>
- <attachment>
  <testimage>/9j/4AAQSkZJRgABAgEBLAEsAAD/7QlMUGhvdG9zaG9wI
  DMuMAA4QklNA+DKUmVzb2x1dG1vbgAA [....]
  AAAQASwAAAABAAEBLAAAAAEAAThCSUDEDRhGWCBHbG9iYWwgTGlnaHRp
  bmcgQWDukQ7Tu VM6LnDklHo7rjEeakDrASSTDf//Z</testimage>
  </attachment>
  </collectionConnection-resultset>
```

Second, the query-parameter 'attachments' can be used to replace the attachment identifier with the mime-encoded contents in the normal resultset. To get multiple attachments, the attachment identifiers need to be separated by commas. For example:

- `http://localhost:9877/action=get&command=search&query=*=*&range=1-10&fields=*&attachments=attachment:4946,attachment:4947`

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

```
- <collectionConnection-resultset>
  <publisher>CollectionConnection Demo</publisher>
  <type>search</type>
  <type>cultural</type>
  <title>search *=*</title>
  <count>6066</count>
  <firstRequested>1</firstRequested>
  <lastRequested>10</lastRequested>
  <sortKey>identifier</sortKey>
- <dc-record>
  <publisher>CollectionConnection Demo</publisher>
  <fixed>Jantje</fixed>
  <identifier>0010C</identifier>
  <title test="1" abc="2924">anis Ian's Society's Child
  (lyrics)</title>
  <filename>sochixx.xxx1</filename>
  <test>Jan 2002</test>
  <testimage>/9j/4AAQSkZJRgABAgEBLAEsAAD/7QlMUGhvdG9zaG9wI
  DMuMAA4QklNA+0KUmVzb2x1dGlvbgAA
  AAAQASwAAAABAAEBLAAAAAEAATh8SUOEDRhGWCBHbG9iYWwgTGlnaHRp
  bmcgQW5nbGUAAAAABAAA
  AHg4QklNBBkSR1ggR2xvYmFsIEFsdGl0dWR1AAAAAAQAAAAe0EJJT0Pz
  C1ByaW50IEZsYWdzAAAA
  CQAAAAAAAAAAAQA4QklNBAo0Q29weXJpZ2h0IEZsYWcAAAAAQAA0EJJ
  TSc0FEphcGFuZXN1IFBy
  aW50IEZsYWdzAAAAAAoAAQAAAAAAAAC0EJJT0P1FONvbG9yIEhhbGZO
  b251IFN1dHRpbmdzAAAA SAAvZmYAAQBsZmYABgAAAAAAAQA
  [...]
```

When "*" is specified in the attachment parameter, all attachments will be included in the resultset.

The function `getAttachment(attachmentID)` in the *CollectionConnection* Client can be used to decode the Mime-data in order to get the original binary data, see Section 8.5.

6.4 Distributing indices with the server

An index can be distributed together with the web site and the browser or full screen browser. The Server contains functionality to make an installation file for this purpose. If a profile is active, three choices are available under the 'file' menu:

- Make server install file
- Make browser install file
- Make full screen browser install file

In essence, these three options carry out the same tasks:

- Copy the active profile
- Copy the index file of the active profile
- Copy the website (denoted by the web-site directory in the settings window)
- Copy the images from the directory specified in the images server
- Copy one of ccServer.exe, ccBrowser.exe, or ccFullscreenbrowser.exe
- Copy WebSvr.dll and ccClient.dll
- When necessary for Z39.50 access copy yaz.dll

Note that the website and image directory are copied with all subdirectories. If the images in the website are to be distributed with the rest, they must reside in or under the website directory. Furthermore, it is important that the website and image server directory do not contain any files that should not be distributed in this way.

The files are compressed and put in an installation file, much like the installation file of *CollectionConnection* itself. The installation file can be used to install a searchable and browsable collection on any computer. Together with the 'list with object identifiers' query and 'record filter script' in the Indexer, this provides a way to distribute an index of a part of the collection (e.g., exhibitions, new acquisitions, etc) easily, both for internal use (information displays in the museum) and for external use.

The *CollectionConnection* Server uses Inno Setup version 4.2.5 that is copyrighted (C) 1998-2001 by Jordan Russell. The *CollectionConnection* installation file itself is also created with Inno Setup.

Distributed Server

7 CollectionConnection Distributed Server

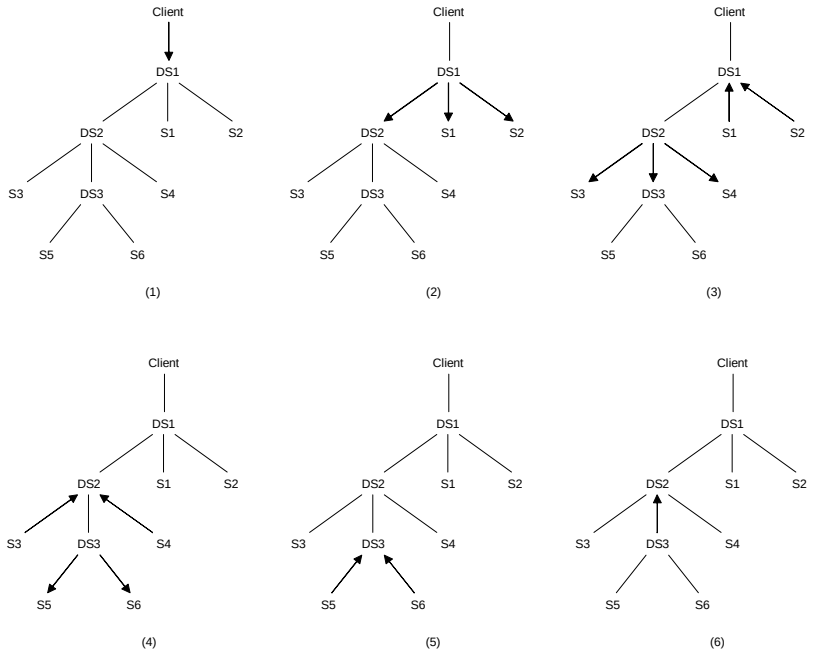
7.1 Introduction

The *CollectionConnection* distributed server is used to search multiple *CollectionConnection* Servers and concatenate the results of the individual servers. The concatenation results are presented as a coherent result-set that is sorted on the given sort-criterion. The Distributed Server is not only able to search normal servers, but also to search other distributed servers. This allows for powerful scalability.

The distributed server is not enabled by default in CollectionConnection. The description is included in the manual for reference purposes. Please contact the reseller for information about purchasing and using the distributed server.

Basically, the Distributed Server is the same as the normal Server. It talks a protocol very similar to the normal server, it's output is XML, and it contains a (HTTP) web server that can either send XML and HTML to the client. The differences are the input (the format of the queries), slight additions to the output, and the settings.

A simple example will clarify the process of distributed searching. This example presumes three distributed servers (DS1, DS2, and DS3) and six normal servers (S1..S6) (Figure 7.1).



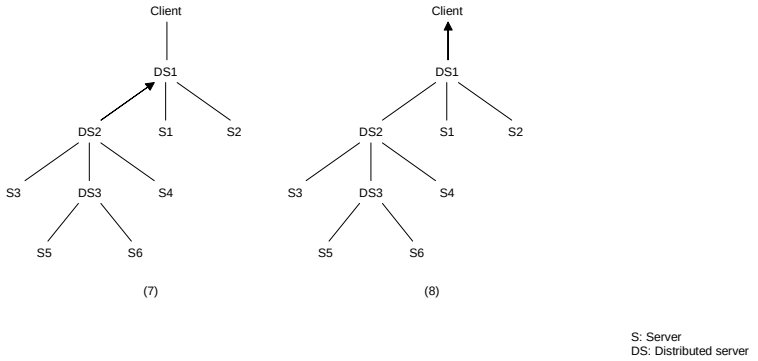


Figure 7.1. Distributed searching in *CollectionConnection*

The process when a client requests information of DS1 is as follows:

1. The client sends the request to DS1.
2. DS1 forwards the request to DS2, S1, and S2
3. S1 and S2 perform the query themselves on their index. The result-sets are sent back to DS1. DS2 forwards the request to S3, S4, and DS3.
4. DS1 has received the results of S1 and S2, but is still waiting for the results of DS2. S3 and S4 send their result to DS2, but DS3 must forward the request to S5 and S6.
5. S5 and S6 send their results back to DS3. DS2 is waiting for DS3, and DS1 is waiting for DS2.
6. DS2 receives the records from DS3. Its results are now complete and it can determine what records will be sent to DS2.
7. DS2 has received now the appropriate records from S3, S4, and DS3, and it can send the records to DS1.
8. Te results of DS1 are now complete, and can be sent to the client.

Please note the following possibilities:

- A normal server can be changed into a distributed server itself. The hierarchy could thus even be more elaborated.
- A client can also directly query (for example) DS2 (thereby the indexes of S1 and S2 would not be queried). In fact, a distributed server does not know whether a client is an end-user of the result-set or a distributed server itself.

7.2 Scalability

As mentioned, the distributed server provides almost unlimited scalability to *CollectionConnection*, because the search process can be performed in a hierarchy of servers. Not only can this be used to split a large database in multiple indexes for performance reasons, it can also be used to provide intra-organizational disclosure of information. For example, a museum can have an index for each warehouse, whereby each warehouse is responsible for its own index. A distributed server can be used to provide transparent access to these indexes as if they were one index. To extend this example, this museum could be collaborating with other museums. Each of these

museums provide their own internet connectivity, but they might want to have a centralized point of access, for example for the whole collection or only for a specific part of it. Again, this can be done easily by using a distributed server; a mere list with (distributed or normal) servers that should be searched by the distributed server is enough.

Some parameters have to be taken into account when using distributed servers:

- **Connection latency**: it costs some time for a computer to connect to another computer. Typically, on a high speed local area network or on the internet backbone, this is under 10 ms. On the internet, this might be as high as 100 ms or more when using transatlantic distribution of servers or with ADSL connections. This time is used for each level in the hierarchy. Steep hierarchies are not a technical necessity; searching of a distributed server is multi-threaded. Rather, hierarchical levels have their origin in organizational causes.
- **Bandwidth**: result-sets must be transferred between computers. Although the result-sets are compressed, it takes time and (sometimes costly) bandwidth. Larger result-sets take of course more time to communicate. Consider for example a 3 layer hierarchy with 4 indexes at each layer. A client connects to a distributed server and requests 10 records. This 'root' distributed server connects to 4 other distributed servers, and each of these connects to 4 normal servers. In this way, 16 indexes are searched. In this example, each of the normal servers sends 10 records to its distributed server (that actually functions as a client to the normal server). Thus, 160 records are already sent. Each of the distributed servers has received 40 records. The distributed servers sort the records and send the 10 highest to the root server. Again, 40 records (4 distributed servers with 10 records each) are sent. The root server sorts them and sends the 10 highest to the initiating client. All in all, 210 records are communicated to send 10 records to the client. This is not a problem in itself, but abuse and uncontrolled distributed searching can of course turn this into a problem.

7.3 Input

The queries on normal servers specify the records that should be contained in the result-set by specifying a starting and ending record. In this way, a client can browse through the result-set without requesting all objects at once. For large result-sets, this is the only way due to limitations in computer memory, processing time, and bandwidth. Because the index is completely controlled by the server, it is possible to specify exact starting and ending records. With a distributed index, this is not possible. Therefore, the input structure is slightly modified for distributed servers. Instead of specifying starting and ending record numbers that should be returned, the client specifies a 'state'. The state depicts for each index in the search hierarchy at what record the server should start searching, whether it should search forward or backward, and how many records should be returned. If the state is left empty by the client, all servers return records from record number 1. The result-set of a distributed server always contains a 'state'-tag that specifies the current state of the result-set (also see the specification of the output in

section 7.4). This tag can be used as input for a next query to the same distributed server. In this way, the client can browse the distributed indexes without worrying about the construction of the state; it is handled by the distributed server itself. This structure has the following parameters that are additional to the normal server:

- `State`: The purpose of the state parameter is discussed above. If the state is left empty, then all servers return results from the first record. The initiating client can also specify what indexes in the hierarchy should be searched. They should be separated by semi-colons. For example: "rmv;tropen".
- `Direction`: This specifies the direction from the current state. It is either "`forward`" or "`backward`".
- `Num`: This parameter specifies the size of the result-set that should be returned.

Example:

```
action=get&command=search&query=*=steen&state=ccdemo:3-
10;sortkeyvalue:ccdemo,00000003,1-
24&direction=forward&num=10&fields=*& sort=identifier
```

7.4 Output

The XML output of the distributed server is the same as the output of the normal server, with two additions: the hierarchical structure with publishers and the state.

7.4.1 Publisher

All servers return in their result-set the publisher of the index. The hierarchical structure is gathered and presented at each level (i.e., at each distributed server). For example:

```
<publisher>
  distributedserver1
   <publisher>server5</publisher>
  <publisher>server6</publisher>
  <publisher>server2</publisher>
  <publisher>
         distributedserver2
         <publisher>server1</publisher>
         <publisher>server3</publisher>
         <publisher>server4</publisher>
  </publisher>
</publisher>
```

This structure shows that the root server is "distributedserver1", and this server queries three normal servers (of which the publishers are server5, server6, and server2) and one distributed server. This latter distributed server ("distributedserver2") queries three normal servers (of publishers server1, server3, and server4). The servers without subnodes are always normal servers; the servers with sub nodes are always distributed servers.

7.4.2 State

As discussed in Section 7.3, the progress of browsing is depicted by the state. From a given state in the result-set, the client can browse either forward or backward. Thus, a query returns the "`state`" tag in the result-set, and this tag is the input for the following query that can either go forward or backward to retrieve respectively the next or previous records. Because the state and direction can be part of an URL if the client is an internet browser, a specific point in the result-set can be bookmarked. Unless the indexes change, the result-set can be reconstructed from the bookmark and browsing can again continue either forward or backward. An example "`state`" is:

```
<state>server5:1-146;server6:1-1;server2:1-6;server1:1-
2;server3:1-95;server4:1-
0;sortkeyvalue:server5,00000001,</state>
```

Note that the "`sortkeyvalue`" in the state is used for browsing backward.

7.5 Settings

The settings of the distributed server are largely similar to the settings of the normal server, but again with some additions. The servers that a query will be forwarded to must be specified. Currently, the server, port, username, and password need to be specified (Figure 7.2). If the distributed server connects to a server, it retrieves the publisher and kind of server (normal or distributed), both of which are shown in the settings pane (Figure 7.3).

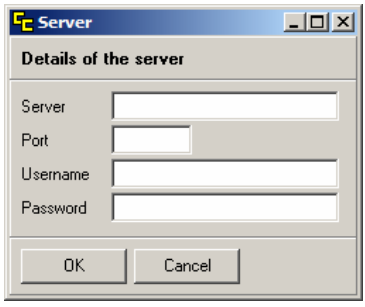


Figure 7.2. Adding a server to search

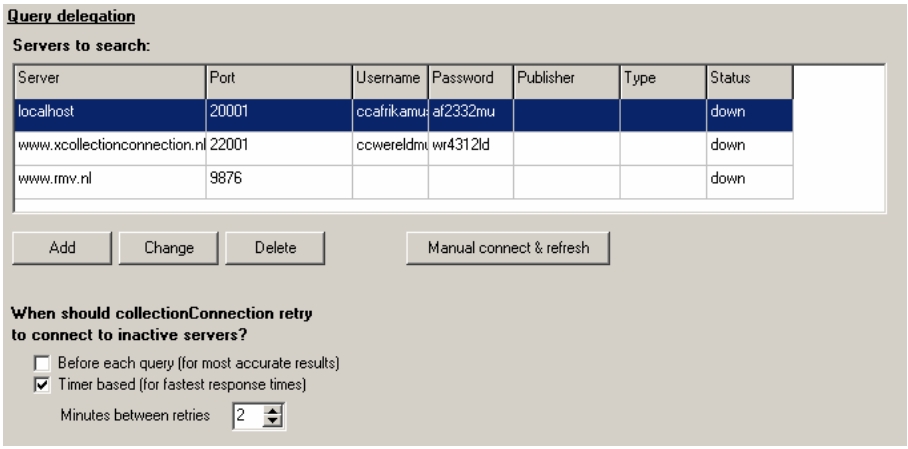


Figure 7.3. Settings of the distributed server

The settings pane also shows whether a server is up or down. In the settings pane it can also be specified what the distributed server should do when a server can not be connected to. The distributed server can (1) try to connect before each query, and (2) try to connect once in a while. The first option ensures up to date results, but connecting to a server that is down takes about one second, so this penalty would be inflicted each query. The second option will try to connect each (configurable) period of time. Both options can also be disabled. In the latter case, the distributed server only connects to other servers once (at startup) and when the button "manual connect & refresh" is pushed. This button can also be pushed in order to refresh the publisher, type, and status of the connected servers.

Client

8 CollectionConnection Client

8.1 Introduction

The *CollectionConnection* Client is an ActiveX component that can be used in Microsoft Windows development environments such as Active Server Pages, Visual Basic, and Borland Delphi. The component (embedded in ccClient.dll) contains a number of objects. These objects send a query to the server, receive the answer, parse the XML structure, and store the answer in memory. The host application of the activeX (e.g., Active Server Pages, or a Delphi, Visual Basic, Visual-C desktop application) can use function calls to request the data one record at a time. There are for example function calls for respectively collections, thesaurus, and keyword search. An example of how the client is used by a host application can be found in Chapter 9 about the Active Server Pages. In addition, the Client contains some utilities to highlight keywords in text, to calculate the aspect-ratio of images, and to send E-Mail. The objects in the activeX will be discussed respectively.

8.2 Generic variables for all objects

The objects of the Client contain two variables that specify the *CollectionConnection* Server that the client talks to:

- `Server` (string)
- `Port` (string)

Furthermore, the client needs to identify itself when authentication on the server is enabled by setting two variables:

- `Username` (string)
- `Password` (string)

If the server is using SSL, the client must also be configured as such. One variable and one procedure are available:

- `UseSSL(boolean)`
- `loadLicense(string)`: procedure with one parameter: the path and filename of the ccLicense file. The certificate in the license file is sent to the server. This is optional; the *CollectionConnection* server currently does not check the validity of the certificate.

If the server resides on the same machine as the client, the server is called `localhost`. The port is specified in the settings window of the Server as the *CollectionConnection* Client Server port. If these variables are not set, the Client tries the default values. The default server is `localhost` and the default port is `7876`.

8.3 ObjectCollections

The objectCollections object can be used to request a list with collections. The following functions are available:

1. `getObjectcollections(first,last).` This function retrieves a number of objectCollections, from the `first` to the `last` collection. With the use of these parameters, records can be retrieved in batches instead of all at once.
2. `Count.` This functions states how much collections are available.
3. `objectCollectionID(rowNumber).` This function retrieves the identifier of a collection in the retrieved set. `Rownumber` must be between `first` and `last`.
4. `objectCollectionName(rowNumber).` This function retrieves the name of a collection. `Rownumber` must be between `first` and `last`.
5. `objectCollectionDescription(rowNumber).` This function retrieves the description of a collection. `Rownumber` must be between `first` and `last`.
6. `objectCollectionCount(rowNumber).` This function retrieves the number of objects that are in a collection.

8.4 ObjectCategories

This object will retrieve the thesaurus. It contains the following functions:

1. `getCategoryChildren(categoryid, first, last).` This function gets the nodes that belong directly under a category, starting at the `first` node and ending with the `last` node.
2. `childCount`. This function specifies the number of nodes that fall under the node as specified by `categoryid`.
3. `categoryName`. The name of the node that is specified by `categoryid`.
4. `childCategoryID(rowNumber).` The Identifier (in hierarchical notation) of the childnode that is specified by `rowNumber`. `RowNumber` must be between `first` and `last`.
5. `childCategoryTerm(rowNumber).` The terms that define the childnode.
6. `childCategoryChildCount(rowNumber).` The number of childnodes of a childnode. (These are grandchildren of the node specified by `categoryid`).
7. `childCategory0bjectCount(rowNumber).` The number of objects that are assigned to a childnode.
8. `categoryParent(rowNumber).` The identifier (in hierarchical notation) of the parent of the current node. This can be used to browse the thesaurus upward.

8.5 ObjectSet

The object 'objectSet' is a collection of objects from the database (in other words: DC-records). It contains the following procedures and variables:

8.5.1 Resultset
The *CollectionConnection* Client provides an easy interface to the XML that is sent by the Server. The following procedures and variables can be used to get the information:

1. procedure
   `get0bjectSet(settype,query,first,last,sortkey,predict,dcl).`
   This function retrieves a number of records from the server. `Settype` is `objectcollection` or `search`. Check section 6.3 for the query format. `First` and `Last` specify the records in the set that must be retrieved. The `sortkey` is

the column on which the results will be sorted. `Predict` specifies the predictor, and `DCL` specifies the Dynamic Content Links that are requested.

2. variable `collectionName`. Only used for the settype "objectcollection". It provides the name of the requested collection.
3. variable `collectionDescription`. Only used for the settype "objectcollection". It provides the description of the requested collection.
4. Function `getTag(rowNumber, tag, attributeSpecifier, occurrence).` This function retrieves the contents of a tag in a given record. Because a XML record can have multiple tags with the same name, the occurrence parameter denotes what tag must be retrieved.
5. Function `tagCount(rowNumber, tag, attributeSpecifier).` This function retrieves the number of tags in a record with the given `attributeSpecifier`.
6. Function `attributeCount(rowNumber, tag, occurrence).` This function returns the number of attributes in a tag.
7. Function `getAttribute(rowNumber, tag, occurrence, attributenumber).` This function can be used to iterate through the attributes of a tag.
8. Function `getAttributeValueByAttributeName(rownum, field, occurrence, attributename).` This function returns the value part of a name-value pair of an attribute. For example, if the field 'title' occurs three times in the second XML-record of the current set, and the third title has as one of it's attributes: `titletype="public"`, then the call `getAttributeValueByAttributeName(2,"title",3,"titletype")` will return "public".
9. Function `getPredictedCount(predictor).` This function returns the size of the resultset of the specified predictor-query (see section 6.3.6 on page 103).
10. Function `processDcl(rowNumber, dclName, Source, Before, After) .` This function highlights all Dynamic Content Links (see section 5.6.13 on page 74) of the type `dclName` in the text of the record. `rowNumber` is the row in the objectSet, `Source` is the text to be rendered, `Before` and `After` are put before and after each occurrence of the "text to search" element of the DCL. The `Before` and `After` can contain the placeholder `#dclvalue#`. This will be replaced with the "link value" element of the current DCL.
    Example (from showObject.asp):
    ```
    curdescription=objectSet.processDcl(rowNum,"thesaurus",curdescr
    iption,"<a
    href=""objectcategories.asp?categoryid=#dclvalue#"">","</a>")
    ```
    When the indexer is properly setup, this function will hyperlink terms from the thesaurus that occur in the description of the record.
11. Function `getAttachment(attachmentID).` This function will return the binary data of the attachment that was specified with the `attachmentID` parameter. The attachment (which is originally Mime encoded by the indexer) is decoded by the client and is returned as a variant array of bytes.

8.5.2 Summarization

CollectionConnection can extract parts of tags where search words occur, see section 6.3.6. This can be used to summarize the search results with large text fields. The context of the first few occurrences of the word in the text is shown, the rest of the text is omitted.

The following properties can be set in the 'objectset'-object (the properties need to be specified before the procedure getObjectSet) is called:

- `summaryThreshold`. This specifies the minimum number of characters that a tag must have before it is summarized.
- `maxSummaryLines`. The maximum number of pieces of text that will be in the summary.
- `optimalSummaryLineLength`. The length of one piece of summarized text. *CollectionConnection* will try to find sentences of this size.
- `maxSummaryLineLength`. If a sentence gets very long, *CollectionConnection* will cut it off at the maximum line length.
- `summaryLineStartText`. The text that is put before each text extract.
- `summaryLineEndText`. The text that is put after each text extract.
- `summaryFields`. The tagnames of tags that should be summarized. This is a comma separated list. Leave it empty to have all fields summarized.

8.5.3 Tags

The `objectSet` object can also request information about the tags that are available in the XML structure as specified by the Indexer. The functions in the client are a wrapper around the `tags` command of the server (please see section 6.3.5 on page 102). The following functions are available:

1. `getTagStructCount`. Get the number of tags in the index.
2. `getTagStructTagName(tagNumber)`. The name of the tag.
3. `getTagStructTagAttributeCount(tagNumber)`. The number of attributes of the tag.
4. `getTagStructTagAttribute(tagNumber, AttributeNumber)`. Get an attribute.
5. `getTagStructTagIndexed(tagNumber)`. If the tag is indexed for full text searching, then this function will return `True`, otherwise it returns `False`.
6. `getTagStructTagParsed(tagNumber)`. If the tag is parsed (split into separate words), then this function will return `True`, otherwise it returns `False`.
7. `getTagStructTagSorted(tagNumber)`. If the is sorted, then this function will return `True`, otherwise it returns `False`.
8. `getTagStructTagDataType(tagNumber)`. This function returns the datatype of the tag (e.g., `string`, `integer`, `datetime`, `script`, `binary`).

The `objectSet` object also contains functionality to create reports of the DC-records. These reports can be visually designed with the report designer. For this, please refer to Chapter 10.

8.6 Spelling

This object is a wrapper around the spelling command of the server (see section 6.3.4 on page 101). The following procedures and functions are available:

1. `getAlternatives(word).` This procedure will get the spelling alternatives for `word` from the server.
2. `Count`. This variable returns the number of alternative words.
3. `Alternative(wordNum).` This function returns the requested alternative.
4. `Score(wordNum).` This function returns the score of the alternative. Alternatives are sorted on the score.

8.7 ThighlightSearchWords

The function `highlight` in the object `THighlightSearchwords` renders HTML in order to put tags around a specific number of words. The words must be specified in the form of a query as described in Section 6.3.6 on page 103. The function has four parameters:

1. `Text`: text to render
2. `Before`: text to put before the to-be-highlighted words
3. `After`: text to put after the to-be-highlighted words
4. `Searchquery`: the query that contains the to be to-be-highlighted words

Example:

```
text=obj.highlight(text,"<font color=""red"">","</font>",query)
```

8.8 ImageSize

The `ImageSize` object is used to determine the height and width of an image on the basis of the maximum height and width. The functionality can be used if the actual height and width of the image are not known, but the image may not exceed a given height and width on a web-page (e.g., for use in a table). The object contains a method and two functions:

1. `setImage(imagename, maxWidth,maxHeight)`. This method loads the image from harddisk, determines the actual width and height, and calculates the target width and height on the basis of `maxWidth` and `maxHeight`. The imagename parameter must refer to an actual file on harddisk including the path, not to the weblocation of an image.
2. `newWidth`: This is the target width of the image.
3. `newHeight`: This is the target height of the image.

Example:

```
<%
   Set img0bj = Server.Create0bject("ccClient.ImageSize")
   img0bj.setImage "c:\wwwroot\1234.jpg",45,45
   verysmallwidth= img0bj.newwidth
   verysmallheight= img0bj.newheight
   Set img0bj = Nothing
%>
<img src="1234.jpg" width="<%=cStr(verysmallwidth)"%>
height="<%=cStr(verysmallheight>">
```

8.9 ImageServerProxy

The `ImageServerProxy` object loads an URL and returns it to the host of the client (e.g., an ASP page). This is useful for use in the image server (see Chapter 11 on page 147), because the image server can run on a port that is denied by a firewall of the user. Next to the `Username`, `Password`, `UseSSL`, and `loadLicense` properties, the `ImageServerProxy` object has one function:

4. `getImage(url)`: This reads the URL and and provides it back to the host as a variant array of bytes. When used in Active Server Pages, the result of the function can be sent to the browser with the `Response.BinaryWrite` procedure, for example:

```
Response.BinaryWrite Img.getImage("http://localhost:9882/img.cci")
```

8.10 Security

The `Security` object contains a function to verify the authenticity of a digitally signed record (see section 6.3.1 on page 99). It contains one function:

1. `verifySignature(record)`: The record should contain both `<dc-record>` tags and a digital signature. The boolean function returns `True` when the signature matches the record. Otherwise, it returns `False`.

8.11 Mail

The `mail` object can be used to send mail. It is a generic object, i.e., it can also be used outside *CollectionConnection*. A discerning feature of other ActiveX mail components is that the *CollectionConnection* Mail object does not need a SMTP-host; if no SMTP-host is specified by the programmer, it will query the DNS-server of the domain in the recipient's email address for MX-records and connect directly to the recipient's mail server. Note that if the recipient's SMTP-host is temporarily down, the object can not send the mail. In addition, some mail servers reject connections from non-trusted IP-ranges. Therefore, it is better practice to specify a trusted SMTP-host manually when the mail object is used, for example the ISP's SMTP-server. The following methods and properties are available:

1. `From`: The name of the sender
2. `FromName`: The email-address of the sender
3. `Subject`: The subject of the email
4. `dnsServer`: The DNS-server. This is used to retrieve the mx-record of the domain of the email-recipient. The DNS-server of a computer can be found in the TCP/IP-Settings of the operating system.
5. `smtpHost`: The address or name of the mail-server.
6. `smtpPort`: The port of the mail-server. The default is 25.
7. `addRecipient`(emailaddress): Add a recipient to which the email will be sent.
8. `addMessageLine`(textline): Add a line of text to the body of the message.

9. `addAttachedFile`(filename,cid): Attach a file to the email. The cid (content identifier) can be used to embed content in an html-based email. The example below shows how.
10. `Send`: This function tries to send the email-message and returns an error messages if something went wrong.
11. `Clear`: This procedure clears all specified information. This procedure must be called if more than one message is sent during the lifespan of the object.

Either the smtpHost or the dnsServer **must** be specified.

Example:

```
Set mail0bj = Server.Create0bject("ccClient.Mail")
mail0bj.From="info@collectionconnection.org"
mail0bj.FromName="CollectionConnection"
mail0bj.addRecipient(request("emailaddr"))
mail0bj.Subject="0bject: "+curTitle
mail0bj.dnsServer=" ns1.collectionconnection.org"
mail0bj.addMessageline("<html><body>This is a test<br>")
mail0bj.addMessageline("<img src=""cid:image1"" border=no>")
mail0bj.addMessageline("</body></html>")
mail0bj.addAttachedFile "c:\temp\image0fBeach.jpg","image1"
sendResult=mail0bj.Send
if sendResult="" then
  response.write("The message was sent successfully")
else
  response.write("There  was  an  error  sending  the  message:
"+sendResult)
end if
Set mail0bj = Nothing
```

Active Server Pages

9 Active Server Pages

9.1 Introduction

Because the communication structure of the client to the server is public, it is easy to create scripts or desktop applications. The Active Server Pages (ASP) that are included in CollectionConnection use the client that is described in Chapter 8 to retrieve and show information from the idex. The look of the pages is easily adjustable. The source code of the scripts is public, so it can be used to study and adapt the ASP. We will first describe the functionality of the pages, after which it will be discussed how the appearance can be adjusted.

9.2 Basic retrieval method

```
' create the CollectionConnection Client object
Set Obj = Server.CreateObject("ccClient.objectSet")

' set the communication parameters
Obj.Server="localhost"
Obj.Port="9876"
Obj.Username="ccdemo"
Obj.password="ccdemopassword"
Obj.loadLicense("c:\program files\collectionconnection
    2.0\demo.serverprofile")
Obj.useSSL=True

' let the client perform the query on the server
Obj.getObjectSet "search","title=green
    grass",1,10,"ccRelevance","","/ "

' show result-set characteristics
Count=Obj.Count

if Count=1 then
  response.write("Found 1 object.<BR>")
else
  response.write("Found "+cStr(Count)+" objects.<BR>")
end if

' iterate throught the results
for j=first to last
  if j<=count then
    ' write the identifier and title for each object
    response.write(objectSet.getTag(j,"identifier","",1))
    response.write(": ")
    response.write(objectSet.getTag(j,"title","",1))
    response.write("<BR>")
  end if
next

Set Obj = Nothing
```

9.3 Interaction process

Interaction between the user, ASP, the client, and the server.

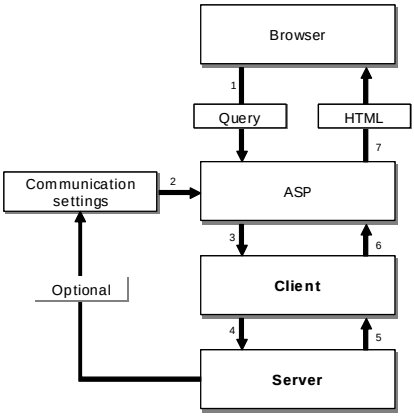


Figure 9.1. Flow of information with ASP pages

1. Determine the query
    a. statically
    b. user input

2. Determine communication settings
    a. Specify manually; in asp page
    b. Use ccsettings.asp
3. Construct call to Client

4. The client sends the query to the server

5. The Client retreives the results

6. In ASP, retrieve the information from the Client

7. The browser receives the result from the webserver

2. Determine the query (user input)
Determine the parameters for communication between the server and the client (static)
a. Server
b. Port
c. Username
d. Password
e. Licens\e location
f. SSL
2. Determine the query (user input)
a. The query itself
b. The range
c. The sort order
d. Predictor queries
e. Dynamic Content Links
3. Check the output

a. Inspect error messages
b. In case of zero results, propose spelling alternatives
4. Iterate the results
a. Determine the first and last record
b. Show the relevant tags
5. Provide navigation possibilities
a. To the relevant Dynamic Content Links
b. To the relevant Predictor results
c. To the previous or next range
d. Narrowing or broadening the search results

9.4 Structure in the included website

Free text search
Browsing tag values
Collections
Collection objects
Thesaurus
Thesaurus objects
Spelling alternatives
Emailing records
Report generator
Binary attachments
Appearance scripts (header, footer, templates)
Miscellaneous scripts (image proxy, mapproxy, linkfunctions, e-commerce, showobject)
verifySignature.asp

10

Reporting

10 CollectionConnection Reporting

10.1 Introduction

CollectionConnection includes a reporting module that can be used to generate reports of search queries. It contains two parts: a standalone program that can be used to visually design reports and a module that creates a report on the basis of a report design and a qeury. The latter module can be used from Active Server Pages. This makes it possible to serve, for example, PDF reports to a webbrowser.

10.2 Creating reports

The report creator is part of the *CollectionConnection* client object `objectSet`. It is implemented as a number of functions:

- `pdfReport`
- `rtfReport`
- `excelReport`
- `textReport`

These functions create respectively Adobe Acrobat, Rich Text Format, Microsoft Excel, and plain text reports. The functions take as parameter the report name (the location on hard disk), e.g.:

```
objectSet.PDFReport("c:\cc\simpleReport.ccr")
```

The functions return the report as an `OLEVARIANT` array of bytes. The active-X host can then stream the report to the client (e.g., in the case of Active Server Pages) or save it to hard disk. A similar set of functions creates the report and saves it directly on the specified location:

- `pdfReportToFile`
- `rtfReportToFile`
- `excelReportToFile`
- `textReportToFile`

These functions need two parameters, the report name and the filename of the to be created report. For example:

```
objectSet.PDFReportToFile("c:\cc\simpleReport.ccr","c:\temp\ccrepo
  rt.pdf")
```

The functions return a string with an error if it occurred.

The `generateReport.asp` script (see Figure 10.1) embeds the provided functionality. It can stream reports to the user, and it can save a report to hard disk, mail it as an attachment to a specified e-mail address (with the *CollectionConnection* mail object), and delete the report from hard disk when it is mailed.

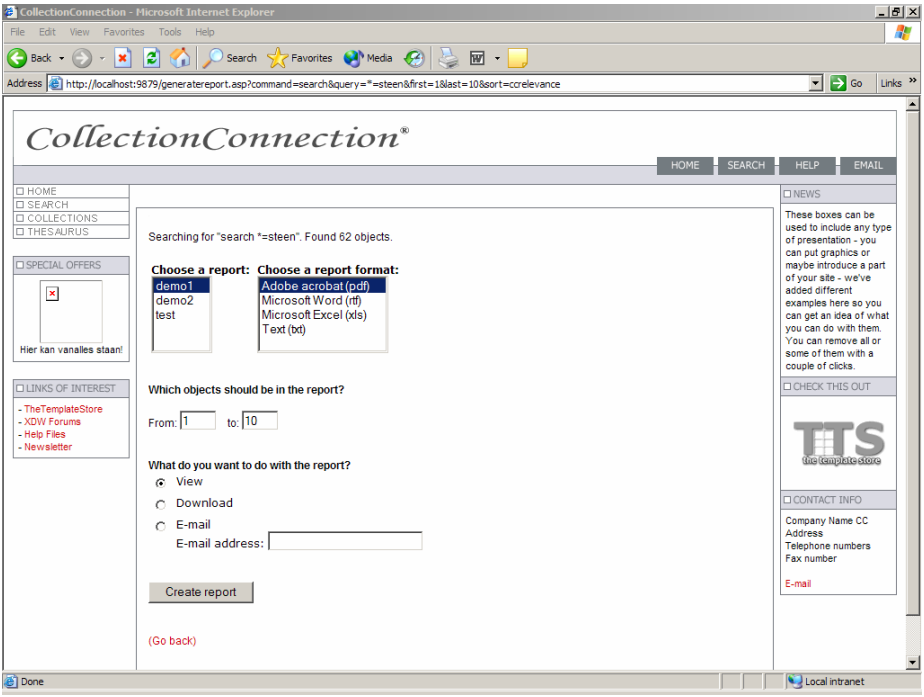


Figure 10.1. generateReport.asp script

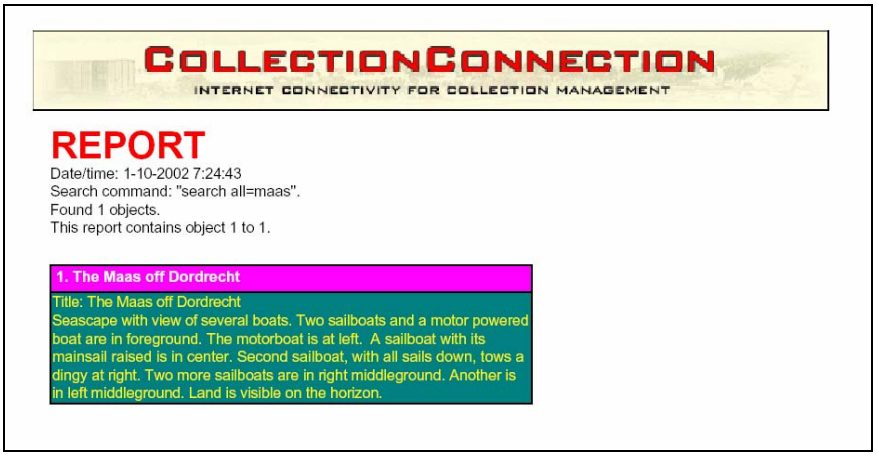

COLLECTIONCONNECTION
INTERNET CONNECTIVITY FOR COLLECTION MANAGEMENT
REPORT
Date/time: 1-10-2002 7:24:43
Search command: "search all=maas".
Found 1 objects.
This report contains object 1 to 1.
1. The Maas off Dordrecht
Title: The Maas off Dordrecht
Seascape with view of several boats. Two sailboats and a motor powered boat are in foreground. The motorboat is at left. A sailboat with its mainsail raised is in center. Second sailboat, with all sails down, tows a dingy at right. Two more sailboats are in right middleground. Another is in left middleground. Land is visible on the horizon.

Figure 10.2. Example pdf-report

10.3 Designing reports

Reports can be visually designed with the report designer. Figure 10.3 shows the screens that comprise the designer. The top pane contains the generic menus and buttons to open, save, and close reports. Furthermore, it contains the report controls. The report controls are the building blocks of the reports. The right pane contains the report

design, and the left pane (the object inspector) shows the properties of a selected object in the report design.

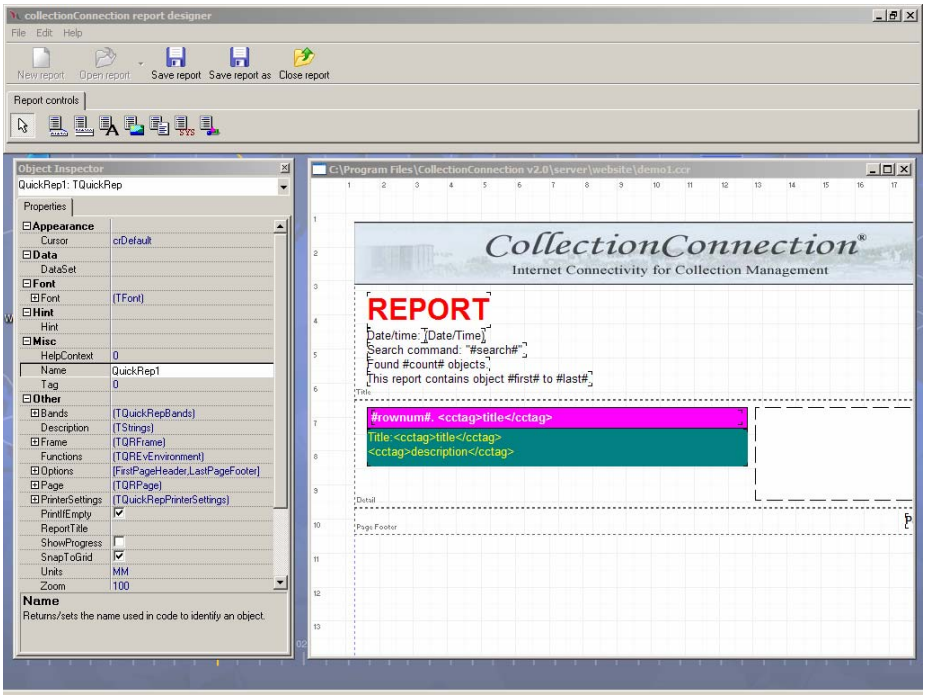


Figure 10.3. Report designer

A report is created by selecting a report control, and clicking somewhere on the report. The control is then dropped on the report with the default properties. Controls can be sized, moved, copied, pasted, and deleted, and the properties can be changed. Important properties will be described for each of the controls.

Band

A band is the basic placeholder for elements of the report. There are several kinds of bands. The kind of band can be set with the property `bandType`. Not all are applicable to *CollectionConnection* reports. The ones that can be used are:

- `Title`. The title is printed once at the start of the report.
- `Detail`. The detail band is printed for each object in the result set. This is where the actual object information is put.
- `Pageheader`. The page header is printed at the top of each page.
- `Pagefooter`. The page footer is printed at the bottom of each page.
- `Columnheader`. The column is printed above the detail band. It can be used to show column headings.

Child band

A child band is a placeholder that is printed under another band or child band. It is just another band but with the property `parentBand`. A child band is printed under it's parent. This can be useful if multiple memo fields must be printed (for the reason of this, please refer to the description of the Memo control).

Label

A label shows a line of text on the report. The `color`, `font`, and `alignment` determine the appearance of the text. The content is specified with the `caption` property. Labels that are put on detail bands can show object information. For this, the caption of the label should specify what DC-field the information should come from. This is done by putting <cctag> and </cctag> around the fieldname. For example, if the caption of a label in the report designer specifies:

```
Title:<cctag>title</cctag>
```

then the report will show the following:

```
This is the title: Orange ball
This is the title: Nice painting
…etc…
```

Labels can also show query specific information. Such information is usually shown in the title band. The following fields can be used:

- `Search`. This shows the query that the report is created from.
- `Count`. The number of objects that is in the result set.
- `First`. The first object in the result set that is shown.
- `Last`. The last object in the result set that is shown.

These fields must be enclosed with #. For example, if the caption of the label specifies:

```
Found #count# objects.
```

then the report will show the following:

```
Found 14 objects.
```

Memo

A label will show only one line of text. A memo can show multiple lines of text. The memo is typically used for fields that can span more than one line, for example the description of a record. Object information is embedded in the same way as with the label, only not in the property `caption` but in the property `lines`. Again, the appearance is specified with, among others, `color`, `font`, and `alignment`. Important other properties of the Memo control are `autoStretch` and `autoSize`. Typically, `autoStretch` is set to `True`, and `autoSize` is set to `False`. Then, the memo is automatically sized vertically to fit the text if the report is created. This is necessary since not all objects will require the same amount of text. If the memo is put at the bottom of a band, it will not overlap with other controls. This is where the child band comes in,

because sometimes several memo fields are needed in a report, but they cannot be put under each other because they are resized when the text is put in and then they possibly overlap. If each memo is put in it's own child band, the band is resized together with the memo and the memo's are aligned vertically without overlapping.

Image

Like it's name suggests, the image shows an image on a report. It can be a static image, for example a logo in the title or page header. The property `Picture` is used to put an image in the control. The button that is shown in the object inspector on this property can be used to browse for an image on harddisk. For images, the properties `stretch` and `autoSize` are important. Images can also be dynamic, to show an image that is specified in a DC-record (typically used on detail bands). For this, the location where the image can be found needs to be specified in the `hint` property of the image. For example:

```
d:\websites\cc.org\thumb\<cctag>image</cctag>.jpg
```

would load images from the directory `d:\websites\cc.org\thumb\` and append the suffix `.jpg` to the contents from the DC-record image field. If the image on hard disk is larger than the image on the report, it is resized automatically, whereby the original aspect-ratio is maintained.

System data

System data is information that is filled in during report creation by the program. There are several kinds of system data (specified by the property `Data`), and the appropriate ones for *CollectionConnection* reports are:

- `Date`
- `Time`
- `DateTime`
- `PageNumber`
- `ReportTitle`

Shape

The shape control is used to put lines, boxes, frames, etc. in the report. The appearance is mostly determined by the kind of shape (with the property `shape`), and with the properties `brush`, `pen`, and `frame`.

Imageserver

11 CollectionConnection Imageserver

11.1 Introduction

The image server is a special purpose HTTP server. It not only serves images, but it can also perform the following server-side manipulations:

- Sizing
- Retrieving a part of the image
- Add a border
- Add a passepartout
- Add a background color
- Add an overlay image
- Add an overlay text

Figure 11.1 shows an example of such manipulations.

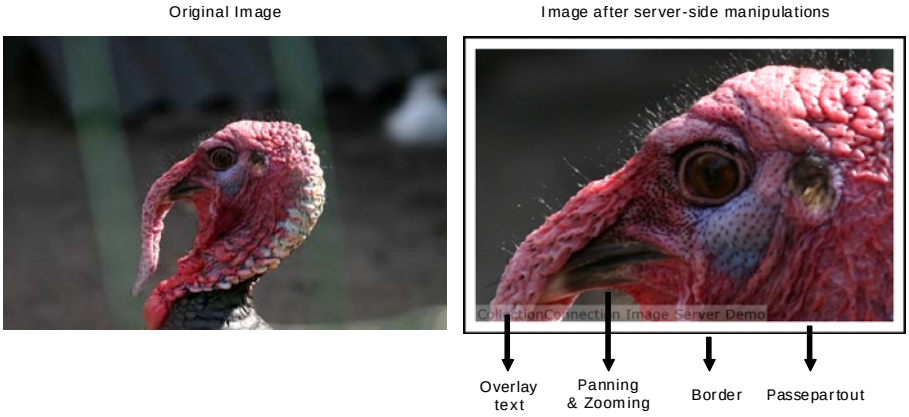


Figure 11.1. Server-side image manipulation

The following sections will respectively discuss how to create *CollectionConnection* images, how to perform server-side manipulations, and how to retrieve the images.

11.2 Imageserver shell integration

11.2.1 Image conversion

In order to be able to perform the server-side operations efficiently, *CollectionConnection* uses a propriety file format. *CollectionConnection* images have the extension `.cci`. Conversion of the images to `.cci` can be done by the server (see section 6.2.12 on page 91) but this will decrease the performance of the server, especially for large images. Coversion can also be done from Windows Explorer. Selecting a `.jpg` or `.jpeg` file and clicking the right mouse button will show a popup menu with the entry 'Convert image to CC'. Selecting a folder and right-clicking will also show a popup menu, now with the entry 'Convert images to CC'. After clicking on the conversion command, the conversion screen appears (see Figure 11.2).

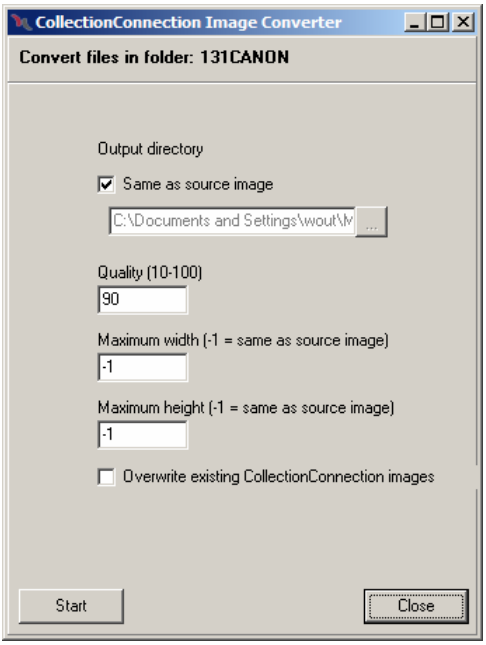


Figure 11.2. Image conversion dialog

The following settings need to be specified:

- `Output directory`. This is the place where the `.cci` images will be stored.
- `Quality`. This is the quality of the resulting `.cci` file. Lower quality results in smaller files.
- `Maximum width` and `maximum height`. This will restrict the width and height of the destination file, which can greatly reduce the `.cci` filesize. It can be useful for internet applications where zooming and panning are not needed and the maximum width and height are limited by the browser anyway.
- `Overwriting existing images`. When this option is enabled, existing `.cci` images will be overwritten. Select this option to overwrite `.cci` images with new settings. Disable this option if only files that have not yet been converted must be converted to the `.cci` format.

After clicking on start, the image conversion starts, and a window shows the progress of the conversion process (Figure 11.3).

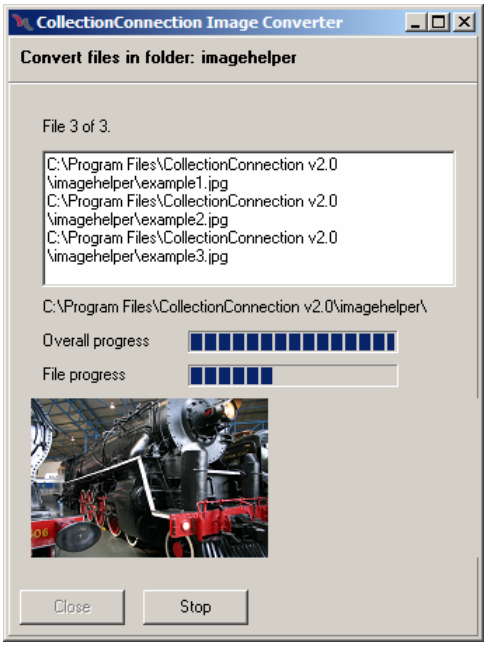


Figure 11.3. Image conversion progress dialog

11.2.2 Image preview

Double-clicking on a `.cci` file in the explorer will open the preview window (see Figure 11.4). The buttons `Previous` and `Next` can be used to browse through the `.cci` images in the current directory.

Figure 11.4. Image preview window

11.3 Image retrieval parameters

The image server is a HTTP server. A request for the image server is specified in a URL. The image server returns a `.jpeg` file. An example request is:

```
http://localhost:9882/IMG_2532.JPG?width=250&height=250&bg=9934ad
```

Figure 11.1 shows some of the elements that the image server can perform on the source image. The parameters for these (which are specified in the URL) will be discussed in the following subsections.

11.3.1 Scaling

Images can be viewed in any size that is requested. The image will automatically be scaled to fit the requested width and height. Furthermore, the server can zoom the image. The following parameters are available for scaling:

- `Width`. The image returned will have exactly this height.
- `Height`. The image returned will have exactly this width.
- `Maxwidth`. The maximum width the image may have. Use this in combination with `maxHeight`.
- `Maxheight`. The maximum height the image may have. Use this in combination with `maxWidth`.
- `X`. Use this to extract a part of the image starting at `X`.
- `Y`. Use this to extract a part of the image starting at `Y`.
- `Zoom`. This parameter can be used to zoom the image. Use it in combination with `X` and `Y`.
- `Bg`. This is the background color of the image. If `width` and `height` are specified but `X`, `Y`, and `Zoom` are not specified, the actual image will be centered and superfluous pixels will get the color as specific with `Bg`. The color specification takes the following syntax: rrggbb, where rr, gg, and bb are the respective hexadecimal components for red, green, and blue. For example, `ff0000` is read background, `ffffff` is a white background, `000000` is a black background, and `9934ad` is a purple background.

11.3.2 Border

The server can put a border around an image. The following parameters are used:

- `Borderwidth`. The width of the border.
- `Borderheight`. The height of the border.
- `Bordercolor`. The color of the border. See under `Bg` how this should be specified.

11.3.3 Passepartout

The server can put a passepartout around an image (some room between the border and the actual image). The following parameters are used:

- `Passepartoutwidth`. The width of the passepartout.
- `Passepartoutheight`. The height of the passepartout.
- `Passepartoutcolor`. The color of the passepartout. See under `Bg` how this should be specified.

11.3.4 Overlay image

The image server can merge the requested image with another image. This can be useful to show copyright or trademark message. The following parameters are used:

- `0verlayfilename`. The name of the image that should be added.
- `0verlayposition`. The position of the added image. This can be:

    c. `N` for north (centered on the top of the image)
    d. `E` for east (centered on the right)
    e. `S` for south (centered on the bottom)
    f. `W` for west (centered on the left)
    g. `NE` for north-east (top-right corner)
    h. `SE` for south-east (bottom-right corner)
    i. `NW` for north-west (top-left corner)
    j. `SW` for south-west (bottom-left corner)

- `0verlayalpha`. To give the added image the appearance of a watermark, the alpha parameter can be specified. It determines how much the image 'shines' through. The value 32 means that the added image is not transparent at all, the value 0 means that the added image is totally transparent.
- `0verlaytransparentcolor`. This parameter is used to specify the transparent area of the added image. The transparent pixels of the added image will not be copied to the requested image. Note that overlay images with a transparent color should be `.gif` files since `.jpg` files do not store color information accurately enough.

11.3.5 Overlay text

The image server can put text on the requested image. This can be useful to show copyright or trademark message. The following parameters are used:

- `0verlaytext`. The text to be shown.
- `0verlaytextalpha`. The amount of transparency with which the text should be shown. See `0verlayalpha` for further explanation.
- `0verlaytextposition`. The position of the text. See `0verlayposition` for further explanation.
- `0verlaytextfontname`. The name of the font to be used.
- `0verlaytextfontsize`. The size of the font to be used.
- `0verlaytextfontcolor`. The color of the font to be used.
- `0verlaytextbackgroundcolor`. To make the text better visible, it can have a background color. See `Bg` for the specification of the format.

11.3.6 Miscellaneous

- `Cache`. When this parameter is specified with `yes`, the `.jpg` image that is created from the `.cci` image will be stored to disk. When the image server gets a request for an image it previously cached to disk, the image will not be recreated from the `.cci` image but it will be served directly from disk. This will improve the performance of the server. Cached images are stored in the same directory as the `.cci` images are in, and their filename starts with `ccImageCache_`. They can safely be deleted, since the server will recreate them when necessary.

11.4 Zoom applet

The *CollectionConnection* imageserver can zoom and pan images. *CollectionConnection* includes a java applet that allows realtime zooming and panning via a webbrowser (see Figure 11.5).

Figure 11.5. Zoom applet

The following HTML-code shows how the applet can be included in a webpage:

```
<APPLET name="cczoomapplet" codebase="http://localhost:9879/"
    archive="imageproxy.asp?server=localhost&port=9882&filename=s
    erverZoom.jar" CODE="serverZoom.class" WIDTH=350 HEIGHT=350>
  <PARAM NAME="picture"
    value="http://localhost:9879/imageproxy.asp?server=localhost&
    port=9882&filename=example1.cci">
  <PARAM NAME="bgcolor" value="ffffff">
  <PARAM NAME="bordercolor" value="444444">
  <PARAM NAME="x" value="0">
  <PARAM NAME="y" value="0">
  <PARAM NAME="basezoom" value="1">
  <PARAM NAME="initzoom" value="1">
  <PARAM NAME="showcontrols" value="yes">
  <PARAM NAME="cacheinitimage" value="no">
</APPLET>
```

The following parameters can be set:

- *Applet tag*. The codebase specifies the location of the applet. The archive specifies the filename of the applet. In the example, the image proxy is used (see section 11.6). The width and height specify the size of the applet.
- *Picture*. This parameter specifies the picture to use. Again, in this example the proxy is used. Note that Java applets for security reasons normally only allow communication to the server that the applet itself is on. Therefore, the image must be on the same server as the applet.
- *Bgcolor*. The color that is used on the background of the applet when the image does not take the whole available space in the applet.
- *Bordercolor*. The color of the border around the applet.
- *x*. The starting x coordinate.
- *y*. The starting y coordinate.
- *Basezoom*. The zoom level that the applet shows when the button is clicked.
- *Initzoom*. The zoom level that is shown when the applet is first started.
- *Showcontrols*. Determines whether the zooming and panning buttons are shown on the applet. The applet can also be controlled from javascript with the following procedures:

```
cczoomapplet.zoomIn();
cczoomapplet.zoomOut();
cczoomapplet.Zoom();
cczoomapplet.Pan();
cczoomapplet.resetImage();
```

  These functions put the applet in the same state as the corresponding buttons shown on the applet itself. An example to use these functions is to call them from the mouseclick event on an image in the browser.
- *Cacheinitimage*. If the initial image is cached, starting the applet will be a bit faster. See also the `cache` setting in section 11.3.6 on page 151.

11.5 Image maps

The image server can serve image maps that are created with the *CollectionConnection* Image Map Editor. For an explanation about image maps and the Image Map Editor, see chapter 12 (section 12.5) on page 168.

An image map consists of some `javascripts`, a `span` definition containing the images, and a `map` section that specifies the hotspots on the image. The image server can both serve the three elements seperately, and it can serve all three at once. The latter is more efficient, but in some cases it might be necessary to include the separate parts in separate sections of a HTML file. There are four functions to get the elements:

- `http://localhost:9882/earth.ccm?mappart=script`: the scripts
- `http://localhost:9882/earth.ccm?mappart=span`: the span
- `http://localhost:9882/earth.ccm?mappart=map`: the map
- `http://localhost:9882/earth.ccm?mappart=html`: the scripts, span, and map together

The last URL is enough to show the map in Internet Explorer. This URL can be used with the *CollectionConnection* Client `ImageServerProxy` object (see section 8.9 on page 128) to include the image map in a web page via ASP, for example:

```
<!--#include file="showobject.asp"-->
<!--#include file="header.asp"-->
Test imagemap!<br>
<%
   Set Img=Server.CreateObject("ccClient.imageServerProxy")
   response.write(img.getImage("http://localhost:9882
     /earth.ccm?mappart=html")
   Set Img=Nothing
%>
<!--#include file="footer.asp"-->
```

The image server can also resize the image map. This can be used to separate the process of creating the image map from the context in which it is published. For that, both the parameters `maxwidth` and `maxheight` need to be added.

- `http://localhost:9882/earth.ccm?mappart=html&maxwidth=700&maxheight=700`

11.6 Image proxy

The image server is a HTTP server for which the port is specified in the main *CollectionConnection* server. Users that are behind a firewall, however, might not be able to reach the image server. The `ImageServerProxy` object in the *CollectionConnection* Client and the `ASP` page `imageproxy.asp` can be used to circumvent this problem. The *CollectionConnection* Client is used to contact the image server directly, and the image will be served to `imageproxy.asp`, that in turn will send it to the user's browser. For this, the *CollectionConnection* Client must not be behind a firewall. `Imageproxy.asp` takes the same parameters as the image server itself with the following additions:

- `Server`. The domain name or IP-address of the image server. If the image server is on the same machine as the *CollectionConnection* Client then `localhost` can be used.
- `Port`. The port of the image server.
- `filename`. The filename of the requested image.

An example request is:

```
http://demo.collectionconnection.nl/imageproxy.asp?server=localhos
t&port=9882&filename=\IMG_2532.JPG&width=250&height=250&bg=9934ad
```

Since `imageproxy.asp` returns an image, it can be used as the source of an image in a HTML page:

```
<img border=0 width=711 height=480
src="/imageproxy.asp?server=localhost&port=9882&cache=yes&filename
=\IMG_2532.JPG&width=711&height=480&bg=ffffff&bordercolor=000000&b
orderwidth=1&borderheight=1&passepartoutcolor=ffffff&passepartoutw
idth=8&passepartoutheight=8">
```

For the same reason as with normal images, there also is a proxy for imagemaps: `mapproxy.asp`. This wil not only load the map itself using the proxy, it will also make sure that the images in the map are loaded via `mapproxy.asp` instead of directly from the server. The following parameters are possible:

- `ccm`. (required). The image map that is requested.
- `Mappart`. (required). The part of the image mapt that is requested (either script, span, map, or html).
- `Maxwidth`. (optional). The maximum widht of the map.
- `Maxheight`. (optional). The maximum height of the map.
- `Proxy`. (required). The ASP page containing the proxy script, usually `mapproxy.asp`.
- `Proxyserver`. (required). The server on which the webserver for `mapproxy.asp` is running.
- `Proxyport`. (required). The port on which the webserver for `mapproxy.asp` is running.
- `Proxyserverssl`. (optional). Specifies whether the webserver uses SSL.
- `Imageserver`. (required). The server on which the image server is running.
- `Imageport`. (required). The port on which the image server is running.
- `Imageserverssl`. (optional). Specifies whether the imageserver uses SSL.
- `Imageusername`. (optional). Specifies the username of the image server when authentication is enabled.
- `Imagepassword`. (optional). Specifies the password of the image server when authentication is enabled.

An example call (with SSL and authentication) is:

```
https://localhost:9879/mapproxy.asp?ccm=earth.ccm&mappart=html&pro
xy=mapproxy.asp&proxyserver=localhost&proxyport=9879&imageserver=l
ocalhost&imageport=9882&maxwidth=700&maxheight=700&imageserverssl=
yes&proxyserverssl=yes&imageusername=demousername&imagepassword=de
mopassword
```

7

Tools

12 Tools

12.1 Introduction

CollectionConnection contains various tools that perform miscellaneous functions. In this chapter, the following tools are described: remote control, the XML viewer, the database transfer program, and the html-imagemap editor.

12.2 Remote Control

The remote control server can be used to view the activity of a remote desktop console and to take over the control of that remote desktop. Furthermore, it provides a chat option between the remote server and the remote client, and a program to transfer files between the remote server and the remote client. Remote control can be useful for remotely assisting a user that is having a problem and to remotely manage or diagnose a computer that is running the *CollectionConnection* server or the *CollectionConnection* indexer.

The remote control server is a standalone program. It is not embedded in the *CollectionConnection* server, because if the *CollectionConnection* server would for some reason be stopped, the remote server would be stopped too, thereby eliminating the possibility to analyze the cause of the interruption and the possibility to start the *CollectionConnection* server again.

12.2.1 Remote Control Server

The remote control server can be run in two ways. First, it can be run as a standalone program. This would mainly be used by a human user that needs assistance. Second, the remote control server can be run as a NT-Service. This option is not available on Windows 98 and ME, only on Windows NT/2000/XP/2003. If the server is configured to run as a service, the server will run without anyone logged in on the computer, and the server will start during system startup. This would mainly be used to perform administrative tasks on remote servers that normally run without user intervention.

Standalone Remote Control Server

The Remote Control Server needs only a few settings (see Figure 12.1):

- Username
- Password
- Port

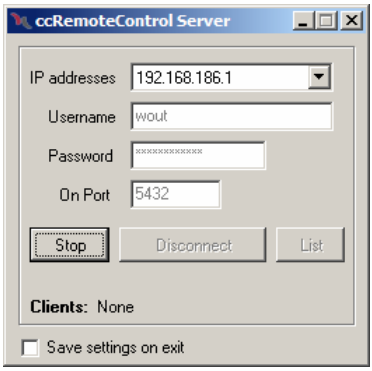


Figure 12.1. Remote Control Server settings

The listbox with IP addresses is shown for remote assistance purposes. When seeking assistance, the helpdesk needs to know the IP address their remote control client should connect to. Note that if the computer is behind a router, the IP addresses listbox probably shows the private IP address rather than the internet IP address.

After pressing start, the remote control server is started and minimized to the system tray. Double clicking on the *CollectionConnection* icon ( ) in the system tray will show the server settings window again. From there, it can be stopped. The current list with connected clients can be viewed by pressing the `List` button. Each connected client can be disconnected individually by pressing the `Disconnect` button and then choosing the client to disconnect.

By checking the `Save Settings` checkbox, the username, password, and port will be saved to harddisk. The next time the remote control server is started, the saved settings will be loaded automatically.

The settings of the remote control server can also be specified with command line parameters. These will override the settings that are saved to disk. The following command line parameters can be used:

- Username
- Password
- Port
- Autostart. The server will automatically be started and minimized to the system tray.

For example:

```
ccRemoteControlServer.exe -username:ccdemo -password:ccdm14 -
  port:1974 -autostart
```

The command lines parameters option can be used in a batch file, in a shortcut, or in an entry in the autorun section of the Windows registry.

Remote Control Service

When the Remote Control Server is run as a service (available on Windows NT, Windows 2000, Windows XP, and Windows 2003) it will start automatically when Windows starts. The service settings can be specified by running `ccRemoteControlService.exe` manually. The following service-related options are available (see Figure 12.2):

- Install as service. If the program is not yet installed as a service, this function will do so.
- Uninstall as service. If the program is installed as a service, this function will uninstall it.
- Start service. If the program is installed as a service but not started, this function will start it.
- Stop service. If the program is installed as a service and started, this function will stop it.

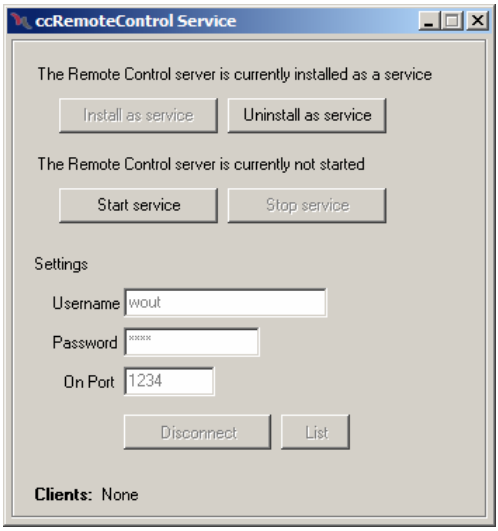


Figure 12.2. Remote Control Server service settings

In addition to the service-related options, the follwing Remote Control Server related options must be specified:

- Username
- Password
- Port

The settings can only be changed when the service is not installed. After changing the settings, the service can be installed again with the new settings.

When started, the remote control service shows the *CollectionConnection* icon ( ) in the system tray on the console. Double clicking on the icon will show the server settings window. From there, it can be stopped by clicking on the X in the title bar. The current list with connected clients can be viewed by clicking on the `List` button. Each connected client can be disconnected individually by pressing the `Disconnect` button and then choosing the client to disconnect.

12.2.2 Remote Control Client

The Remote Control Client has three functions. First, it can be used to take control of the desktop of the Remote Control Server. Second, it can be used to chat with a user that is using the computer where the Remote Control Server is running on, and third, it can be used to transfer files.

Settings

To connect to a Remote Service, the following settings need to be specified (see Figure 12.3):

- Username
- Password
- Port

By checking the `Save Settings` checkbox, the username, password, and port will be saved to harddisk. The next time the remote control client is started, the saved settings will be loaded automatically.

The settings of the remote control server can also be specified with command line parameters. These will override the settings that are saved to disk. For example:

```
ccRemoteControlClient.exe -server:192.168.2.1 -username:ccdemo -
  password:ccdm14 -port:1974
```

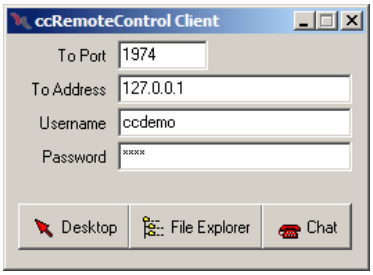


Figure 12.3. Remote Control Client settings

When the settings are specified, the three buttons `Desktop`, `File Explorer`, and `Chat` start their respective options. Each will be discussed below.

File explorer
The file explorer can be used to transfer files between the client and the server. The current directory of both the local computer and the server are in the left pane. The contents of the current directories are in the right pane. Files can be transferred by selecting a file or directory and clicking `Send (upload)` or `Get (Download)`.

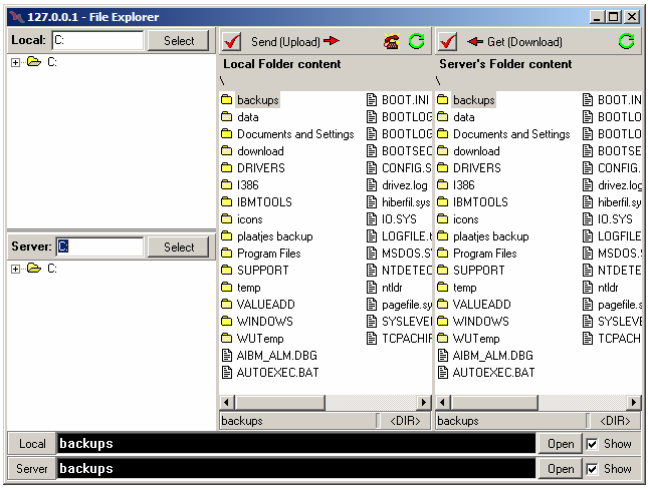


Figure 12.4. Remote Control File Explorer

Chat

With the Chat option (Figure 12.5), text messages can be exchanged between the user of the client and the user of the server.

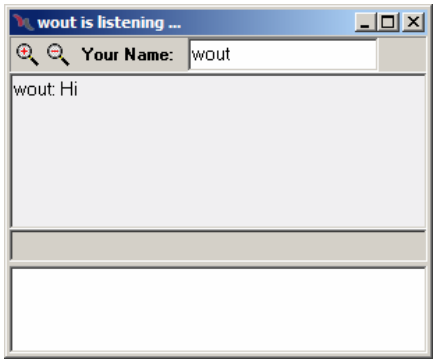


Figure 12.5. Remote Control Chat

Remote desktop

The Remote Desktop provides the main functionality of the Remote Control Client. It shows the desktop of the server and will send mouse and keyboard actions of the client to the server (see Figure 12.6).

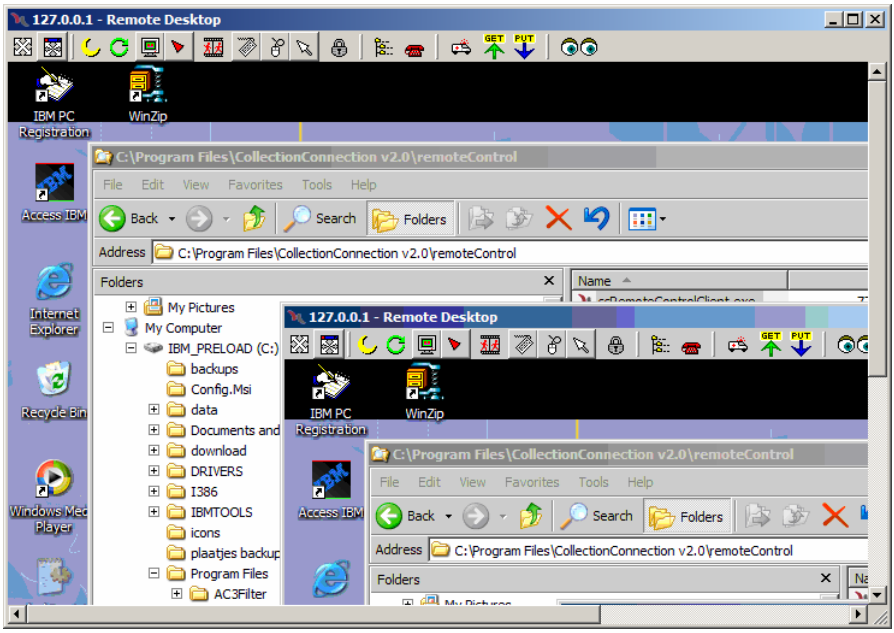


Figure 12.6. Remote Control Desktop

When started, it displays the remote server desktop. The following options are available:

- Run the desktop client full-screen.
- Scale the desktop to the current size of the client window.
- Enter passive mode. The toolbar will be made invisible.
- Refresh the desktop.
- Automatically refresh the desktop.
- Show the position of the mouse of the server on the desktop screen.
- Use the GDI-hook. This will improve the performance on the server-side. For this, the file `RemHook.dll` must be available in the directory of the executable of the Remote Control Server. Some programs bypass the GDI so use this option with care. Pressing the `refresh` button will ensure that the whole screen is sent.
- When this is enabled, keyboard events of the Remote Control Client are sent to the server.
- When this is enabled, mouse events of the Remote Control Client are sent to the server.
- Pressing this button will enable keyboard events, mouse events, showing the server mouse cursor position, and automatically refreshing the desktop. This will provide full control with one mouseclick.
- This will lock input from the server. Keyboard and mouse actions on the server side are now disabled.
- This will start the File Explorer.
- This will start the Chat window.
- This will simulate pressing <ctrl>-<alt>-<del> on the server.
-

This will copy the clipboard from the server to the clipboard from the client. This only works with text.
-

This will send the client's clipboard to the server's clipboard. This only works with text.
- This will show a menu with all available options:
    a. To "GDI Hook" mode. See button.
    b. Colors. The color depth can be specified. Lower color depth means better performance.
    c. Block size. The block size can be specified. A larger block size can increase performance on fast network connections.
    d. Frame rate. The number of times per second that the desktop is refreshed when 'automatically refreshing the desktop' is enabled.

e. Hide Wallpaper. On low speed network connections, hiding the wallpaper of the server can improve performance.
f. To passive mode. See button.
g. New File Explorer. See button.
h. New Chat. See button.
i. Start Autorefresh. See button.
j. Show server mouse. See button.
k. Lock server input. See button.
l. Close. This will close the Remote Desktop.

12.3 XML Viewer

The program `ccXMLViewerV20.exe` can be used to view the output of an XML-request to a *CollectionConnection* server. It can both show the formatted XML (using the Windows Explorer internally) and the source XML (see Figure 12.7).

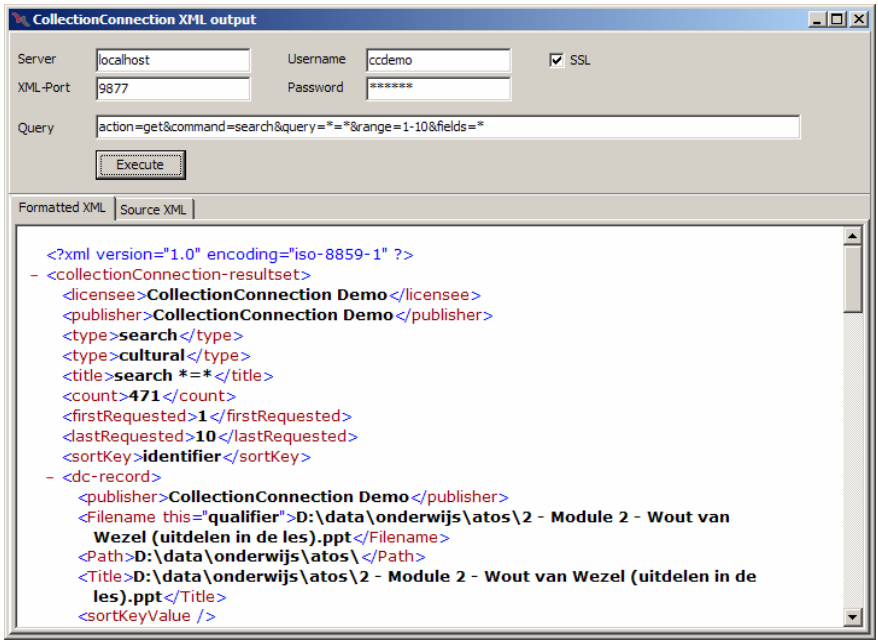


Figure 12.7. XML Viewer

The following settings must be specified:

- *Server*. The ip-address or domain name of the server that will provide the data. If the server is on the same machine as the database transfer program, the server is called `localhost`.
- *XML-port*. The port on which the server provides the XML-HTTP protocol (see section 6.2.8 on page 86).
- *Username*. When the server has enabled authentication for the XML-HTTP protocol, the username should be specified.

- *Password*. When the server has enabled authentication for the XML-HTTP protocol, the password should be specified.
- *SSL*. SSL should be enabled when the server has enabled SSL for the XML-HTTP protocol.
- *Query*. The query that specifies the data that is requested from the server. Use `*=*` for all records. For further information, see section 6.3.6 on page 103.

For binary fields, the server returns encoded XML (see section 5.6.11.2 on page 69 and section 6.3.9 on page 108). The XML-viewer will recognize such binary fields and show the `Save Attachment` button (see Figure 12.8). Using this button, the attachment can be decoded and saved to disk.

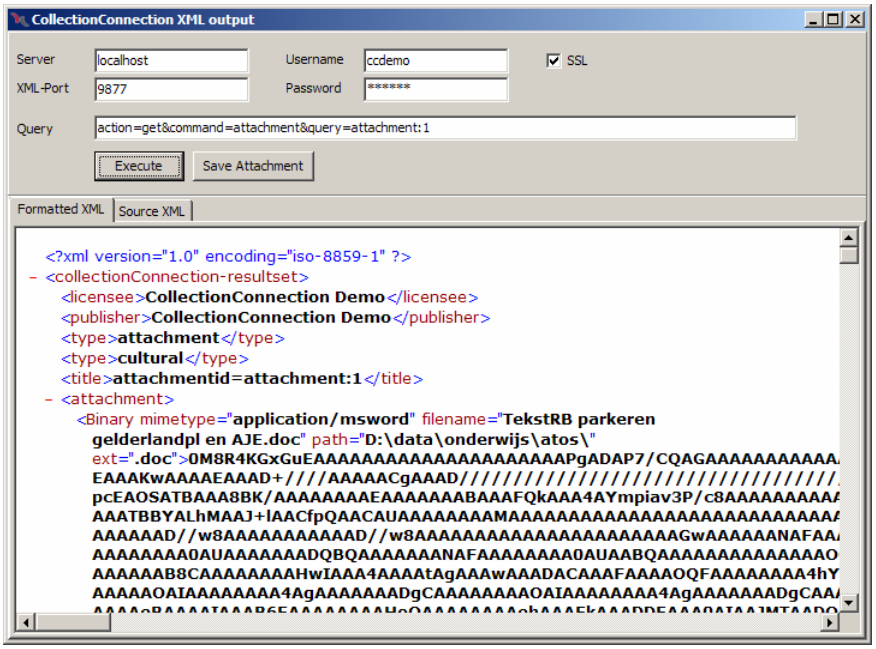


Figure 12.8. XML Viewer – saving binary attachments

12.4 Database Transfer

With the database transfer program `cc2dbv20.exe`, data from a *CollectionConnection* index can be exported to a regular relational database. The database transfer connects to a *CollectionConnection* server, reads the requested information, and stores it in a database. The database transfer program uses the *CollectionConnection* client; it should be installed on the computer the program is used.

The following settings specify where the database transfer program gets its data (see also Figure 12.9):

- *Server*. The ip-address or domain name of the server that will provide the data. If the server is on the same machine as the database transfer program, the server is called `localhost`.
- *CC-port*. The port on which the server provides the *CollectionConnection* protocol (see section 6.2.7 on page 85).
- *Username*. When the server has enabled authentication for the *CollectionConnection* protocol, the username should be specified.
- *Password*. When the server has enabled authentication for the *CollectionConnection* protocol, the password should be specified.
- *SSL*. SSL should be enabled when the server has enabled SSL for the *CollectionConnection* protocol.
- *Query*. The query that specifies the data that is requested from the server. Use `*=*` for all records. For further information, see section 6.3.6 on page 103.

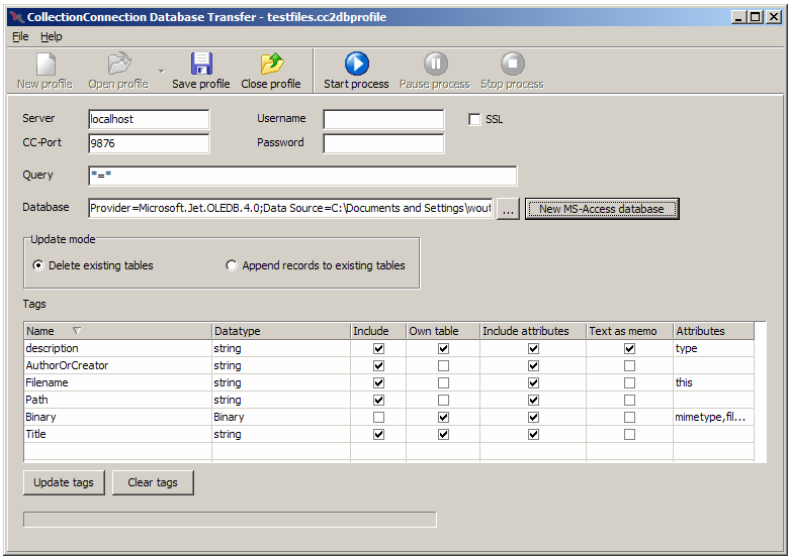


Figure 12.9. Database Transfer

Figure 12.10 shows the resulting database structure of the settings in Figure 12.9. Each tag can either be put in the main table `Objects`, or get it's own table (e.g., in Figure 12.10 `title` and `description`). Creating a table for a tag is necessary when the tag can occur multiple times in an XML-record. The tables are linked by the field `objectid`; each `objectid` identifies an XML-record. Furthermore, tables for tags contain a field with a serial number. Attributes can also be stored in the tables as separate fields. Their fieldnames will be `tagname_attr_attributename`, for example: `Filename_attr_this` for the attribute `this` in the tag `Filename`.

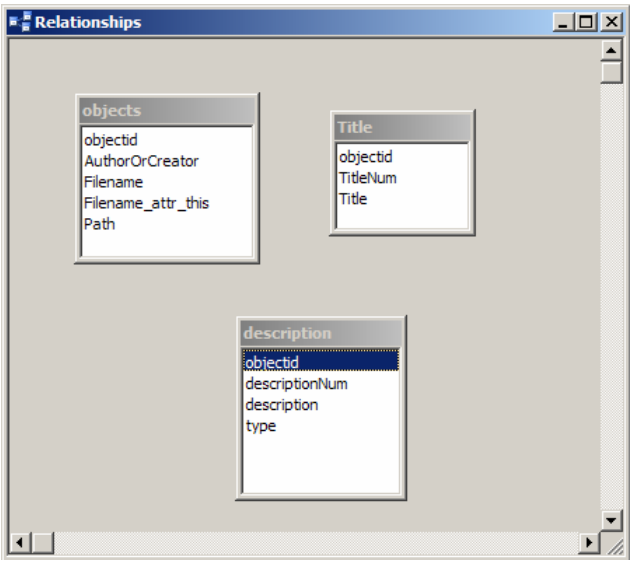


Figure 12.10. Resulting database structure (from MS-Access)

The following settings specify where the database transfer program will store its data:

- *Database*. The destination of the data is an ADO-database. The ADO-connection can be specified by pressing the … button. Alternatively, a new Microsoft Access database can be created after which the ADO-connection specification is filled in automatically.
- *Update mode*. If tables where the data will be stored already exists, they can either be deleted before the transfer process starts, or records can be appended to these tables.
- *Tags*. Each tag in the Index can be put in the database. By pressing the `udpate tags` button, the database transfer program will connect to the *CollectionConnection* Server and request the tag structure. The program will try to keep the current tag settings as they are. To clear the current settings, press the `Clear tags` button. The matrix shows the tag information (name, datatype, attributes) and the following configurable transfer settings:

    m. *Include*. This setting determines whether the tag will be included in the database or not.
    n. *Own table*. The tag can be put in the `objects` table (for 1:1 relations) or in its own table (for 1:n relations).
    o. *Include attributes*. Attributes can be included in the database.
    p. *Text as memo*. Text will be truncated to a maximum of 255 characters in text fields. To include all text, the tag can be stored as a memo field.

12.5 Image map editor

With the image map editor (`ccImageMapEditor.exe`), html-imagemaps can be created. These are images for use in webbrowsers with areas that respond to mouse movements and mouseclicks. Figure 12.11 shows an example. The normal image (left in Figure 12.11) shows the map of the world. Suppose we want to highlight continents when the mouse hovers over a continent. Furthermore, we want the text of the continent to be shown red then. The other way around should also work; when the mouse moves over the continent text, the text must be shown red and the continent must be highlighted. When the user clicks on a continent or on the continent text, a checkmark must be shown next to the text of the continent to indicate it was clicked (shown in the right example in Figure 12.11). After the continent is clicked, the browser should show a specific file in another frame. In the right example in Figure 12.11, the mouse was clicked on Australia, after which the user moved the mouse cursor over the content South America (note that the mouse cursor itself is not shown in the figure).

Figure 12.11. Image map editor example

To create an example such as in Figure 12.11, there are multiple source images needed:

- *Normal image*. The image that is shown by default.
- *Mouse-over image*. This image will show when the mouse is moving over an area.
- *Mouse-down image*. This will be shown when the mouse-button is down. This can be used to simulate a button-click.
- *Mouse-clicked image*. This image is used to specify what should be shown when the area is clicked on.

If an event will not result in a change in the image, the corresponding image is not needed either. The image map editor links areas on the images to mouse-events. For example, when the mouse is over a certain part of the image, not the whole mouse-over image will be shown; only the part that is specified in the image map editor. The same goes for the mouse-down and mouse-clicked images.

The four images that are needed to create the image map in Figure 12.11 are shown in Figure 12.12. In the image map editor, the continents need to be marked. Figure 12.13 shows the main screen of the image map editor program. Each continent, each text, and each checkmark button is marked as an area. For each area (or hotspot) on the image, the highlighting and other actions must be specified (see also Figure 12.14, Figure 12.15, and Figure 12.16). For example, consider the area that marks Europe. When the mouse is over this hotspot, not the area of the normal image must be shown, but of the mouse-over image. Furthermore the text Europe must be colored red, i.e., the area that contains the text should also be show from the mouse-over image instead of from the

normal image. And when the user clicks on the hotspot, a checkmark must be shown next to the text; the part of the mouse-clicked image marking the button next to the text must be shown. All such settings are easy to specify with the image map editor.

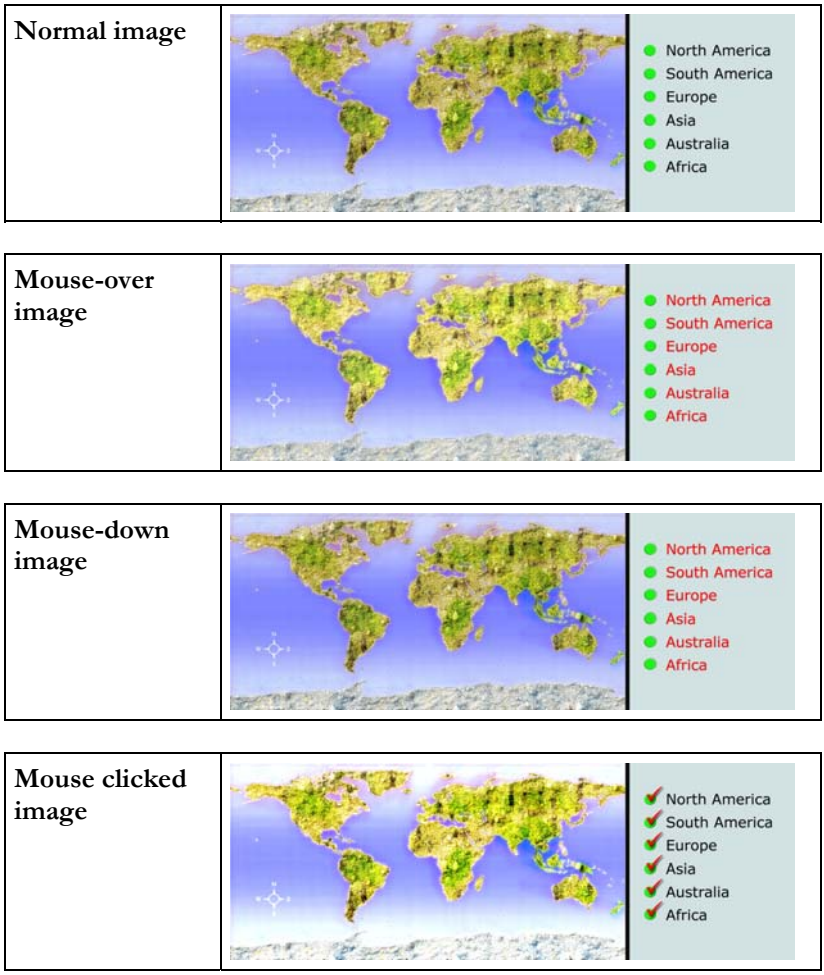


Figure 12.12. Image map editor source images

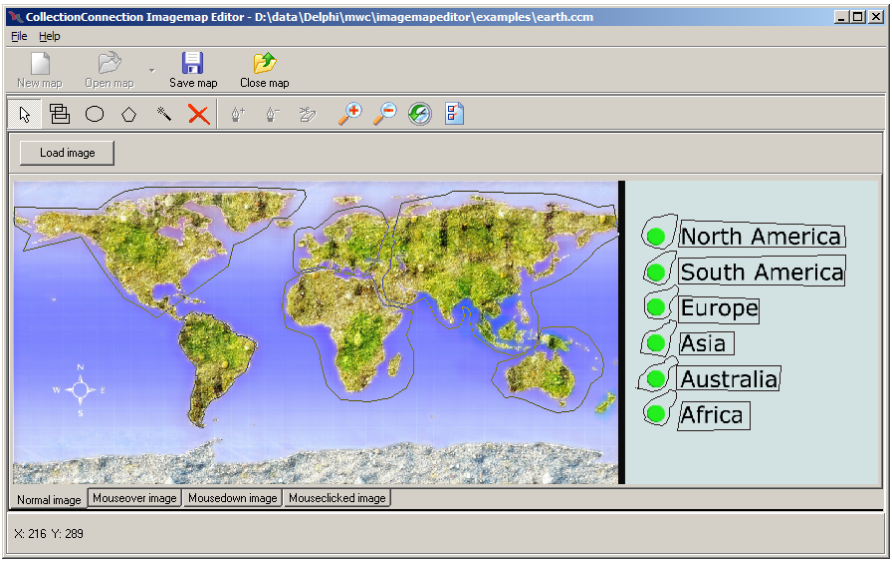
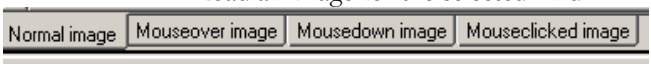


Figure 12.13. Image map editor – editing hotspots

The following tools are available to mark areas (also called hotspots):

- Default mode. You are able to select hotspots.
- Create a new rectangular hotstpot.
- Create a new elliptical hotspot.
- Create a new polygonal hotspot.
- Create a new hotspot with the magic wand. The statusbar will show the magic wand settings for accuracy and tolerance. The hotspot will be converted to a polygonal hotspot that can be edited manually.
- Delete the selected hotspot.
- Add a point to a polygonal hotspot.
- Delete a point from a polygonal hotspot.
- Delete a line from a polygonal hotspot.
- Zoom in on the image. Hotspots can be placed with more accuracy.
- Zoom the image out. Note that zooming out and then zooming in again will possibly distort the allocation of hotspots!
- Reset the zoom.
- Edit the properties of the selected hotspot.

-

Load an image for the selected kind.
- 

The image that is shown.

There are three kinds of properties for a hotspot:

- *Generic properties*. The hotpots need a name. Furthermore, they can show a hint (a tooltip baloon when the mouse is over the hotspot in the browser), and a text in the statusbar of the browser when the mouse is over the hotspot (Figure 12.14).
- *Hyperlink properties*. The hyperlink specifies what happens upon clicking the hotspot. The target frame specifies in what frame the URL of the hyperlink will be activated (Figure 12.15).
- *Hovering effects*. The hovering effects specify what happens when the user moves the mouse over the hotspot or clicks on the hotspot (Figure 12.16). Here it can be specified what other hotspots should be highlighted when the mouse moves over the hotspot or what areas should act like they have been clicked on.

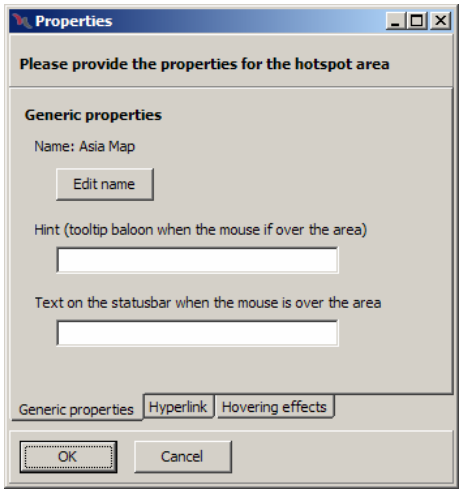


Figure 12.14. Image map editor hotspot properties – generic

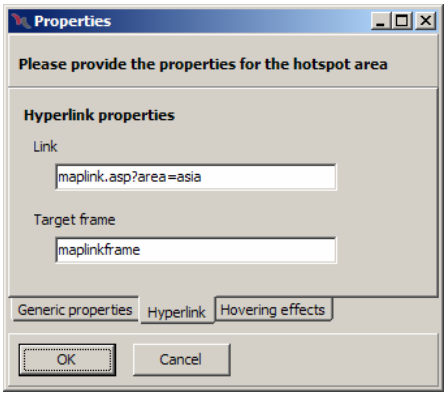


Figure 12.15. Image map editor hotspot properties - hyperlink

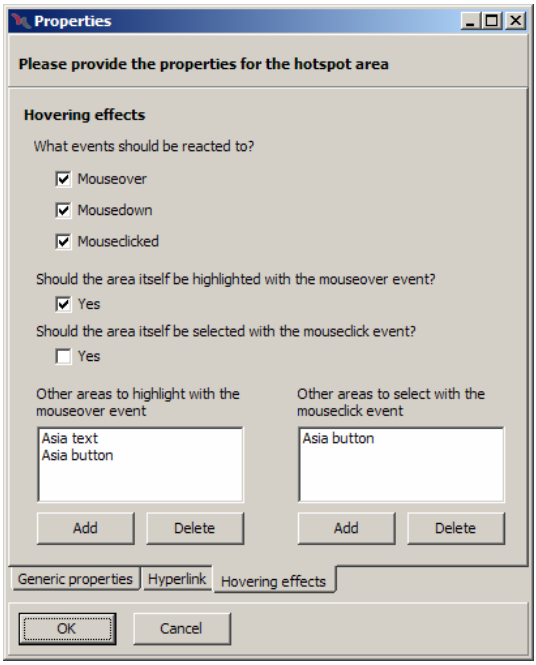
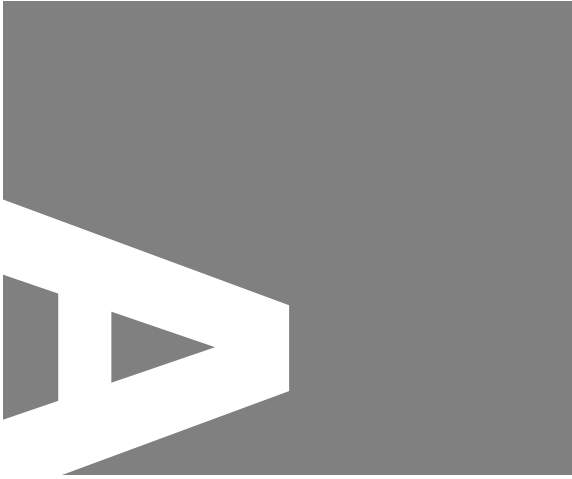


Figure 12.16. Image map editor hotspot properties – hovering effects

Image maps are saved in one file that is both used by the image map editor to edit it and for showing the image map in the browser with the use of the *CollectionConnection* image server. The default installation of *CollectionConnection* contains an example image map (earth.ccm). In section 11.5 on page 153, it is explained how image maps can be shown directly from a browser or using Active Server Pages.



Appendix A: Scripting reference

Appendix A Scripting Reference

A.1 Basic syntax

CollectionConnection executes scripts written in the syntax of the Basic programming language. This syntax supports:

- `sub .. end` and `function .. end` declarations
- `byref` and `dim` directives
- `if .. then .. else .. end` constructor
- `for .. to .. step .. next` constructor
- `do .. while .. loop` and `do .. loop .. while` constructors
- `do .. until .. loop` and `do .. loop .. until` constructors
- `^ * / and + - or <> >= <= = > < div mod xor shl shr` operators
- `try .. except` and `try .. finally` blocks
- `select case .. end select` constructor
- array constructors (`x:=[ 1, 2, 3 ];`)
- `exit` statement
- access to object properties and methods (`0bjectName.Sub0bject.Property`)

A.2 Script structure

The script structure is made of two major blocks: a) function and sub declarations and b) the main block. Both are optional, but at least one should be present in the script. Some examples:

```
SCRIPT 1:
SUB DoSomething
  CallSomething
END SUB

CallSomethingElse


SCRIPT 2:
CallSomethingElse


SCRIPT 3:
FUNCTI0N MyFunction
  MyFunction = "0k!"
END FUNCTI0N
```

Statements in a single line can be separated by ":" character.

A.3 Identifiers

Identifier names in script (variable names, function and procedure names, etc.) follow the most common rules in Basic. They should begin with a character (a..z or A..Z), or '_', and can be followed by alphanumeric characters or the '_' character. Identifiers names can not contain any other character or spaces.

Valid identifiers:

```
VarName
_Some
V1A2
_____Some____
```

Invalid identifiers:

```
2Var
My Name
Some-more
This,is,not,valid
```

A.4 Assign statements

Assign statements (assign a value or expression result to a variable or object property) are built using "=". Examples:

```
MyVar = 2
Button.Caption = "This " + "is ok."
```

A.5 Character strings

Strings (sequence of characters) are created in Basic using double quote (") character. Some examples:

```
A = "This is a text"
Str = "Text "+"concat"
```

A.6 Comments

Comments can be inserted inside script. You can use the single quote (') character or `REM`. Comments will finish at the end of the line. Examples:

```
 ' This is a comment before ShowMessage
ShowMessage("0k")

REM This is another comment
ShowMessage("More ok!")

' And this is a comment
' with two lines
ShowMessage("End of okays")
```

A.7 Variables

There is no need to declare variable types in script. Thus, you declare variable just using `DIM` directive and its name. A compile error will be raised if a variable is used but not declared in a script. Examples:

```
SCRIPT 1:
SUB Msg
  DIM S
  S = "Hello world!"
  ShowMessage(S)
```

```
END SUB

SCRIPT 2:
DIM A
A = 0
A = A+1
ShowMessage(A)
```

A.8 Indexes

Strings, arrays, and array properties can be indexed using "[" and "]" characters. For example, if Str is a string variable, the expression Str[3] returns the third character in the string denoted by Str, while Str[I + 1] returns the character immediately after the one indexed by I. More examples:

```
MyChar = MyStr[2]
MyStr[1] = "A"
MyArray[1,2] = 1530
Lines.Strings[2] = "Some text"
```

A.9 Arrays

Scripts support array constructors and support for variant arrays. To construct an array, use "[" and "]" characters. You can construct multi-index array nesting. You can then access arrays using indexes. If an array is multi-index, separate indexes using "," . If a variable is a variant array, the script automatically supports indexing in that variable. A variable is a variant array if it was assigned using an array constructor or if it was created using VarArrayCreate procedure. Arrays in scripts are 0-based. Some examples:

```
NewArray = [ 2,4,6,8 ]
Num = NewArray[1] //Num receives "4"
MultiArray = [ ["green","red","blue"] , ["apple","orange","lemon"]
]
Str = MultiArray[0,2] //Str receives 'blue'
MultiArray[1,1] = "new orange"
```

A.10 If statements

There are two forms of if statement: `if...then..end if` and `if...then...else..end if`. Like normal basic, if the if expression is true, the statements are executed. If there is an else part and the expression is false, then the statements after else are executed.
Examples:

```
IF J <> 0 THEN Result = I/J END IF
IF J = 0 THEN Exit ELSE Result := I/J END IF
IF J <> 0 THEN
  Result = I/J
  Count = Count + 1
ELSE
  Done = True
END IF
```

A.11 while statements

A while statement is used to repeat statements, as long as a control condition (expression) is evaluated as true. The control condition is evaluated before the statements. Hence, if the control condition is false at the first iteration, the statement sequence is never executed. The while statement executes its constituent statement repeatedly, testing the expression before each iteration. Examples:

```
WHILE (Data[I] <> X) I = I + 1 END WHILE
WHILE (I > 0)
  IF Odd(I) THEN Z = Z * X END IF
  X = Sqr(X)
END WHILE

WHILE (not Eof(InputFile))
  Readln(InputFile, Line)
  Process(Line)
END WHILE
```

A.12 loop statements

Scripts support loop statements. The possible syntaxes are:

```
DO WHILE expr statements LOOP
DO UNTIL expr statements LOOP
DO statements LOOP WHILE expr
DO statement LOOP UNTIL expr
```

Statements will be execute WHILE expr is true, or UNTIL expr is true. if expr is put before statements, then the control condition will be tested before the iteration. Otherwise, the control condition will be tested after the iteration. Examples:

```
DO
  K = I mod J
  I = J
J = K
LOOP UNTIL J = 0

DO UNTIL I >= 0
  Write("Enter a value (0..9): ")
  Readln(I)
LOOP

DO
  K = I mod J
  I = J
J = K
LOOP WHILE J <> 0

DO WHILE I < 0
  Write("Enter a value (0..9): ")
  Readln(I)
LOOP
```

A.13 for statements

Scripts support for statements with the the following syntax:

```
FOR counter = initialValue TO finalValue STEP stepValue statements
NEXT
```

A for statement sets the counter to initialValue, repeats execution of statements until "next" and increments the value of the counter by stepValue, until the counter reaches finalValue. The step part is optional, and if omitted stepValue is considered 1. Examples:

```
SCRIPT 1:
FOR c = 1 TO 10 STEP 2
  a = a + c
NEXT

SCRIPT 2:
FOR I = a TO b
  j = i ^ 2
sum = sum + j
NEXT
```

A.14 select case statements

Scripts support select case statements with the following syntax:

```
SELECT CASE selectorExpression
  CASE caseexpr1
  statement1
  …
CASE caseexprn
  statementn
CASE ELSE
  elsestatement
END SELECT
```

if selectorExpression matches the result of one of caseexprn expressions, the respective statements will be executed. Otherwise, elsestatement will be executed. The Else part of case statement is optional. Example:

```
 SELECT CASE uppercase(Fruit)
  CASE "lime" ShowMessage("green")
  CASE "orange"
  ShowMessage("orange")
  CASE "apple" ShowMessage("red")
CASE ELSE
  ShowMessage("black")
END SELECT
```

A.15 function and sub declaration

Declarations of functions and subs are similar to basic. In functions to return function values, use the implicited declared variable which has the same name of the function. Parameters by reference can also be used, using BYREF directive. Some examples:

```
SUB HelloWord
  ShowMessage("Hello world!")
END SUB

SUB UpcaseMessage(Msg)
  ShowMessage(Uppercase(Msg))
END SUB

FUNCTION TodayAsString
  TodayAsString = DateToStr(Date)
```

```
END FUNCTION

FUNCTION Max(A,B)
  IF A>B THEN
    MAX = A
  ELSE
    MAX = B
  END IF
END FUNCTION

SUB SwapValues(BYREF A, B)
  DIM TEMP
  TEMP = A
  A = B
  B = TEMP
END SUB
```

A.16 available predefined functions

Showline
Scanf
Etc.

Appendix B: Copyright notices and disclaimers

Appendix B Copyright notices and disclaimers

Copyright © 2004 VanWezel Informatiesystemen. CollectionConnection is a registered trademark. All rights reserved.

DISCLAIMER: THE SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT SHALL VANWEZEL INFORMATIESYSTEMEN OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF VANWEZEL INFORMATIESYSTEMEN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Installation procedures Copyright © 1997-2004 by Jordan Russell, http://www.jrsoftware.org/.

CollectionConnection uses the destructor.de XML Parser.

Portions of this software are Copyright © 1993 - 2003, Chad Z. Hower (Kudzu) and the Indy Pit Crew - http://www.nevrona.com/Indy/. For these portions, the following conditions hold:

> Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
>
> Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation, about box and/or other materials provided with the distribution.
>
> No personal names or organizations names associated with the Indy project may be used to endorse or promote products derived from this software without specific prior written permission of the specific individual or organization.
>
> THIS SOFTWARE IS PROVIDED BY Chad Z. Hower (Kudzu) and the Indy Pit Crew "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Z39.50 connectivity is provided by YAZ of Index Data. For this part of the software, the following conditions hold:

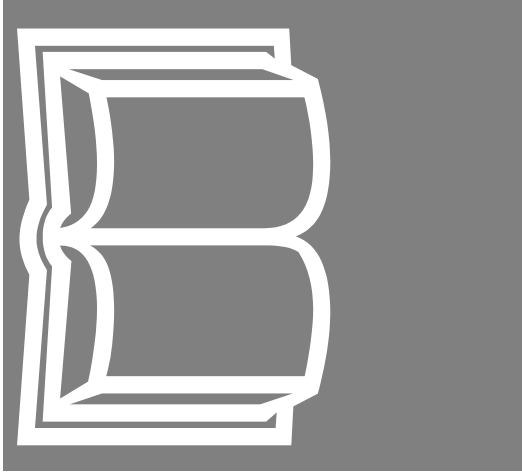
Copyright © 1995-2004 Index Data ApS.

Permission to use, copy, modify, distribute, and sell this software and its documentation, in whole or in part, for any purpose, is hereby granted, provided that:

1. This copyright and permission notice appear in all copies of the software and its documentation. Notices of copyright or attribution which appear at the beginning of any file must remain unchanged.

2. The names of Index Data or the individual authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED, OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL INDEX DATA BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.



Index

Index

A

ActiveX component 123
addAttachedFile (client Mail procedure) 129
addMessageLine (client Mail procedure) 128
addRecipient (client Mail procedure) 128
Adlib specific settings 53
ADO Database connection 39
ADO/Oracle specific settings 39
After (client THighlightSearchWords parameter) 127
Alternative (client Spelling procedure) 127
Architecture 21
ASP server 87
Attachment command 109
Attachment, query parameter 109
attributeCount (client ObjectSet procedure) 125
Attributes 72
Authentication 107

B

Batch process index creation 38

Before (client THighlightSearchWords parameter) 127
browser, server as 79
Browsetag 103

C

Caching by the server 84
Categorychildren 101
categoryName (client ObjectCategories procedure) 124
Categoryobjects 101
categoryParent (client ObjectCategories procedure) 124
ccClient.dll 123
childCategoryChildCount (client ObjectCategories procedure) 124
childCategoryID (client ObjectCategories procedure) 124
childCategoryObjectCount (client ObjectCategories procedure) 124
childCategoryTerm (client ObjectCategories procedure) 124
childCount (client ObjectCategories procedure) 124
Clear (client Mail procedure) 129
Client 123
Collection membership 107
CollectionConnection Client 123
collectionDescription (client ObjectSet procedure) 125
Collectiongrouplist 100
Collectionlist 100
collectionName (client ObjectSet procedure) 125
Collections, querying 99
Combine indexes 37
Conditions editor 47
Conjunction 107
Core components of CollectionConnection 21
Count (client objectCollections procedure) 124
Count (client Spelling procedure) 127
Creating a new indexer profile 31

D

Database structures 41
Database Transfer 166, 168
Datatype 69
Digital signature (full text search parameter) 104
Direction (distributed server query parameter) 118
Disjunction 107
Distributed search query parameters 108
Distributed server 115
Distributed server settings 119
Distributing indices with the server 110
dnsServer (client Mail procedure) 128
Dynamic Content Links 74

E

Export 166

F

Fields editor 43
Files, indexing of 54
Filter specification for indexing files 55
Filters for file indexing 56
From (client Mail procedure) 128
FromName (client Mail procedure) 128
Full text search 103
fullscreen browser, server as 80

G

getAlternatives (client Spelling procedure) 127
getAttachment (client ObjectSet procedure) 125
getAttribute (client ObjectSet procedure) 125
getAttributeValueByAttributeName (client ObjectSet procedure) 125
getCategoryChildren (client ObjectCategories procedure) 124
getImage (client ImageServerProxy procedure) 128
getObjectcollections (client objectCollections procedure) 124
getObjectSet (client ObjectSet procedure) 124
getPredictedCount (client ObjectSet procedure) 125
getTag (client ObjectSet procedure) 125
getTagStructCount (client ObjectSet procedure) 126
getTagStructTagAttribute (client ObjectSet procedure) 126
getTagStructTagAttributeCount (client ObjectSet procedure) 126
getTagStructTagDataType (client ObjectSet procedure) 126
getTagStructTagIndexed (client ObjectSet procedure) 126
getTagStructTagName (client ObjectSet procedure) 126
getTagStructTagParsed (client ObjectSet procedure) 126
getTagStructTagSorted (client ObjectSet procedure) 126
Getting started 9
Global scripting variables 35

H

highlight (client THighlightSearchWords procedure) 127
HTTP/HTML-ASP 87
HTTP/XML port 86

I

Image map editor 168
image proxy 154
Image server settings 91
ImageServerProxy (client object) 128
ImageSize (client object) 127
Index data structure 102
Index distribution 110
Index postprocessing script 35
Index preprocessing script 35

Index record postprocessing script 35
Index tag script 35
Index transmission 97
Indexer 31
Indexing options 71
Indexing Outlook Email messages 62, 63
Indexing Outlook Express Email messages 62
Indexing structured files 58
Installation 27

L

Loading a serverprofile 81
loadLicense (client procedure) 123

M

Mail (client object) 128
mapproxy.asp 154

N

Negation 106
Nesting 107
newHeight (client ImageSize procedure) 127
newWidth (client ImageSize procedure) 127
NT-Service, server as 83
Num (distributed server query parameter) 118

O

OAI 90
ObjectCategories (client object) 124
Objectcollection 100
objectCollectionCount (client objectCollections procedure) 124
objectCollectionDescription (client objectCollections procedure) 124
objectCollectionID (client objectCollections procedure) 124
objectCollectionName (client objectCollections procedure) 124
ObjectCollections (client object) 123
ObjectSet (client object) 124
Open Archives Initiative 90
Opening an Indexer profile 32
Operators editor 45
Oracle Database connection 40
Outlook Email messages, indexing of 62, 63
Outlook Express Email messages, indexing of 62
Output of queries 108

P

Password (client variable) 123
Port (client variable) 123
Postprocessing script 35
Predict, full text search parameter 104
Preprocessing script 35
Procedures and functions 35
Process flow of index creation 23
processDcl (client ObjectSet procedure) 125
Publisher 83

Q

Queries in the indexer 48
Query delegation (distributed server) 119
Query designer 50
Query output 108
Query parameters of the server 98
Query, full text search parameter 103

R

Record postprocessing script 35
Remote log server 84
Remote log viewer 85
Resultset (client ObjectSet) 124

S

Score (client Spelling procedure) 127
Scripting in tags 66
Scripting in the Indexer 35
Search engine spider 93
Searchquery (client THighlightSearchWords parameter) 127
Send (client Mail procedure) 129
Sending the index 33
Server 79
Server (client variable) 123
Server profile, creating 80
Server settings 83
Service, server as 83
setImage (client ImageSize procedure) 127
smtpHost (client Mail procedure) 128
smtpPort (client Mail procedure) 128
Spell 101
Spelling (client object) 126
Spelling alternatives 101
Spider 93
State (distributed server query parameter) 118
Stopwords 73

Structured files 58
Subject (client Mail procedure) 128
Summarization (objectset properties) 125
Summarizing large texts 104

T

Tag datatype 69
Tag script 35
Tag source 65
tagCount (client ObjectSet procedure) 125
Tags 64, 102
Tags (client ObjectSet) 126
Text (client THighlightSearchWords parameter) 127
Thesaurus node membership 107
Thesaurus, querying 100
THighlightSearchWords (client object) (client object) 127

U

Username (client variable) 123
UseSSL (client variable) 123

W

Web crawler gateway 93
Webserver 87
Windows Scripting Host 88

X

XML files, indexing of 53
XML Viewer 165

Z

Z39.50 server 96
Zoom applet 152